
1 WS-Security policy profile of 2 WS-PolicyConstraints

3 Working Draft 03, 28 June 2005

4 **Authors:**

5 Anne Anderson (anne.anderson@sun.com)

6 **File name:**

7 ws-security-profile-of-ws-policy-constraints

8 **Abstract:**

9 This document defines predicates for specifying constraints on the message security domain
10 covered by the OASIS WS-Security Standard. These predicates are expressed using the
11 generic policy constraint language WS-PolicyConstraints, which is based on the OASIS
12 eXtensible Access Control Language (XACML) Standard. By expressing constraints using this
13 generic constraint language, any policy processor for WS-PolicyConstraints can verify a
14 message against a WS-Security policy, and can automatically find a mutually acceptable WS-
15 Security policy based on the individual policies of two or more parties. No plug-ins or
16 modifications to the policy processor for WS-PolicyConstraints are required for handling this or
17 any other domain's policy constraints.

18
19 The predicates defined in this document are alternatives to those specified in WS-SecurityPolicy.
20 To enable an easy comparison between the two languages, this document has been organized
21 according to the Assertions defined in WS-SecurityPolicy. The advantages of using WS-
22 PolicyConstraints rather than WS-SecurityPolicy for expressing WS-Security policies are that
23 messages using WS-Security can be automatically verified as to conformance with a specified
24 policy, and that two parties, such as a consumer or client service and a producer or server
25 service, can automatically negotiate a mutually acceptable WS-Security policy for
26 communicating between them. It means that new types of security policy constraints can be
27 added without having to upgrade or replace the policy processor.

28 **Status:**

29 This version of the specification is a working draft.

30 Table of Contents

31	1 Introduction (non-normative).....	3
32	1.1 Notation.....	5
33	2 WS-Security.....	6
34	3 Policies about WS-Security.....	7
35	4 WS-SecurityPolicy.....	8
36	5 Using WS-PolicyConstraints for WS-Security Policies.....	9
37	5.1 Multiple constraints on a single nodeset.....	9
38	5.2 Limited XPath expressions.....	10
39	6 Actual WS-Security policy predicates.....	11
40	6.1 Specification version.....	11
41	6.2 Security tokens.....	12
42	6.3 Integrity Assertion.....	15
43	6.4 Confidentiality Assertion.....	17
44	6.5 Visibility Assertion.....	17
45	6.6 Security Header Assertion.....	18
46	6.7 MessageAge Assertion.....	19
47	7 Lessons learned and future work.....	20
48	7.1 XPath intersections.....	20
49	7.2 New functions.....	20
50	7.3 Constraints on the message processor.....	20
51	7.4 Overly constrained policies.....	21
52	7.5 Cost and value of abstraction.....	21
53	8 References.....	22
54	Revision History.....	23
55	Notices.....	24
56		

1 Introduction (*non-normative*)

57

58 The policy framework currently expressed by *WS-Policy* [WSP] requires the definition of policy
59 “Assertions” (predicates) for each domain to which policy is to be applied. Three examples of
60 specifications defining such Assertions have been published to date:

- 61 • *WS-PolicyAssertions* [WSPA], defining some general-purpose Assertions,
- 62 • *WS-SecurityPolicy* [WSSP], defining policy Assertions for WS-Security [WSS] and other specifications
63 that might cover the same message security space, and
- 64 • *WS-ReliabilityPolicy* [WSRP], defining policy Assertions for WS-Reliability [WSR] and other
65 specifications that might cover the same reliable messaging space.

66 Each of these sets of Assertions is specific to its domain. The syntax of each Assertion is domain-
67 specific, and the ways in which instances of the Assertions are to be interpreted, matched, and verified
68 are described in the domain-specific specification text. In order to handle each Assertion, a policy
69 processor must incorporate a domain-specific code module that understands the interpretation of that
70 Assertion as defined in the domain-specific specification. The interpretation is subject to human error, so
71 without strict conformance tests, different implementations of the processor for each Assertion may not
72 be consistent.

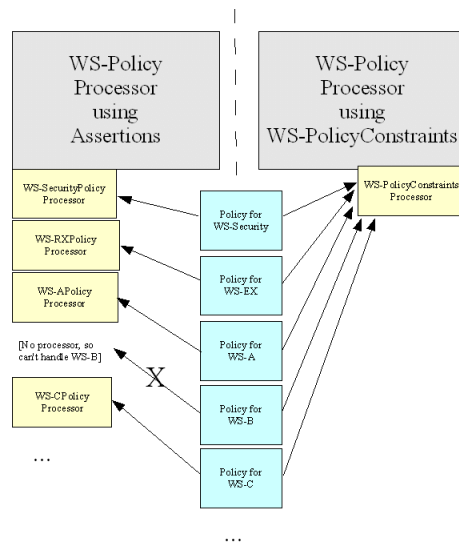
73 As policies are used with more and more domains, the number of domain-specific Assertion modules
74 that must be supported in each policy processor will increase, along with the possibility of interpretation
75 errors, version mismatches, and missing modules. If a customer defines a new type of policy for a new
76 application, the customer must arrange to have modules added to every policy processor for handling
77 the Assertions used the new policy type. It is important to understand that the *WS-Policy* Assertion
78 model requires that each policy processor be configured with code to recognize and implement each
79 Assertion in each domain with which that policy processor will be used.

80 While the authors of *WS-Policy* suggest that, in the future, policy Assertions should be defined as part of
81 the specification to which these policy Assertions apply, this serves only to reduce the total number of
82 specifications. The implementation of each specification must still include a new module that can handle
83 the Assertions defined in the specification.

84 An alternative to the Assertions model is specified in *WS-PolicyConstraints* [WSPC]. In this model, a
85 generic language for specifying policy predicates, or “constraints”, is defined. This generic language is
86 based on the *OASIS eXtensible Access Control Markup Language* [XACML] functions, as used in the
87 *XACML Profile for web-services* [WSPL]. Expressions in this standard language can then be used to
88 express policy predicates for any domain. Any policy processor that supports the generic language can
89 understand, match, and verify any policy written in the generic language, removing the need to have new
90 code modules for each new type of policy Assertion.

91 The following diagram illustrates this difference.

92



93 In addition to enabling the use of generic policy processors, the *WS-PolicyConstraints* language provides
 94 another benefit to web services policy, derived from XACML: policy predicates can be evaluated directly
 95 against a message to confirm that the message conforms to the policy. This is because XACML function
 96 arguments can consist of XPath expressions to be evaluated against actual messages, which could be
 97 SOAP or other types of messages. For example, assume a policy writer wants to constrain the
 98 acceptable values for a username in the

99 `"/S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:UserName"`
 100 element of a SOAP message. The policy writer wants to limit the acceptable values to be names that
 101 start with the string "Zoe". Using *WS-PolicyConstraints*, the policy writer can specify that the value
 102 obtained by evaluating the XPath expression
 103 `"/S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:Username/text`
 104 `()` against actual SOAP messages must match the regular expression string "Zoe.*" using the standard
 105 XACML "string-regex-match" function. In order to constrain the acceptable values for some other
 106 element of the SOAP message, the policy writer can use the same function with different XPath
 107 expression and regular expression arguments. Using *WS-SecurityPolicy* however, the policy writer must
 108 express this policy using an instance of the new "SecurityToken" element defined in *WS-*
 109 *SecurityPolicy*. The policy writer must correctly use the new "SecurityToken" element, its "Claims"
 110 element, its "SubjectName" element, its "MatchType" XML attribute, and the values for the
 111 "MatchType" attribute, as defined in the *WS-SecurityPolicy* specification. Likewise, the policy processor
 112 must contain a domain-specific module for *WS-SecurityPolicy* that recognizes and correctly interprets
 113 each of these elements. This module must know that the value contained in the *WS-SecurityPolicy*
 114 "SubjectName" element is a regular expression that must be matched against the value contained in
 115 the `"/S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:Username"`
 116 element, even though there is no direct reference to this element in the policy: it is specified only in the
 117 text of the *WS-SecurityPolicy* specification. The implementation of all this is specific to the "Username"
 118 element: the *WS-PolicyConstraints* "Username" Assertion can't be used to make regular expression
 119 matches against other elements of a message.

120 By using predicates that can refer to message elements directly, *WS-PolicyConstraints* greatly reduces
 121 the number of new elements that must be defined to express policy information. With the current *WS-*
 122 *Policy* Assertions model, however, a new element must be defined for each component of a message for
 123 which policy is to be specified.

124 This document illustrates how the alternative *WS-PolicyConstraints* model could be used to express the
 125 Assertions defined in *WS-SecurityPolicy*. It is intended to serve as a proof-of-concept for *WS-*
 126 *PolicyConstraints*, as well as providing examples for the use of *WS-PolicyConstraints* that may be of help
 127 to policy developers for other domains.

128 It is important to recognize that *WS-PolicyConstraints* can be used with the *WS-Policy* framework, or with
 129 any other policy framework that addresses the same level of concerns addressed by *WS-Policy*, that is,

130 any framework that defines how to express combinations of constraints to compose a policy.

131 **1.1 Notation**

132 The following XML Internal Entities are used to make the examples more compact and easier to read:

```
133 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" />  
134 <!ENTITY xfunc "urn:oasis:names:tc:xacml:1.0:function:" />  
135 <!ENTITY xdata "urn:oasis:names:tc:xacml:1.0:data-type:" />  
136 <!ENTITY x509 "...the URI of the Web Services Security X.509  
137 Certificate Token Profile..." />
```

138 The following namespace identifiers are used:

```
139 wsse http://schemas.xmlsoap.org/ws/2002/12/secext  
140 ds http://www.w3.org/2000/09/xmldsig#  
141 xenc http://www.w3.org/2001/04/xmlenc#  
142 wsu http://schemas.xmlsoap.org/ws/2002/07/utility  
143 wsp http://schemas.xmlsoap.org/ws/2002/12/policy  
144 xsd http://www.w3.org/2001/XMLSchema  
145 wspc ...a new URI to be defined for WS-PolicyConstraints...  
146 sp ...a new namespace for security policy elements
```

2 WS-Security

147

148 This section serves as a brief introduction to *WS-Security*.

149 *WS-Security* is a set of SOAP [SOAP] extensions that “provides three main mechanisms: ability to send
150 security tokens as part of a message, message integrity, and message confidentiality... These
151 mechanisms can be used independently (e.g. to pass a security token) or in a tightly coupled manner...”
152 [WSS Lines 125-132]. The extensions are added to the SOAP envelope header as part of a new
153 `<wsse:Security>` element.

154 The contents of this `<wsse:Security>` element can include some one or more of the following element
155 types, each of which might occur more than once:

- 156 1. A generic ID and reference mechanism: this can be used with other types defined either in *WS-*
157 *Security* or in other specifications to associate identifiers with elements, and to then reference those
158 identified elements from elsewhere in the `<wsse:Security>` header; a generic ID may also be used
159 in the `<Body>` of the SOAP message envelope,
- 160 2. `<wsse:UsernameToken>`: contains a username security token defined in the *WS-Security*
161 specification, and extended in the *Web Services Security Username Token Profile 1.0*,
- 162 3. `<wsse:BinarySecurityToken>`: contains a binary security token defined in the *WS-Security*
163 specification. There are specific subtypes defined for various X509 token types in the *Web Services*
164 *Security X.509 Certificate Token Profile*. There are specific subtypes defined for Kerberos token
165 types in the *Web Services Security: Kerberos Token Profile (draft)*.
- 166 4. `<ds:SignedInfo>`: contains a signature conforming to the *XML Digital Signature* specification
167 [XDS],
- 168 5. `<xenc:ReferenceList>`: contains a manifest conforming to the *XML Encryption* specification
169 [XENC],
- 170 6. `<xenc:EncryptedKey>`: contains an encrypted key conforming to the *XML Encryption*
171 specification,
- 172 7. `<wsu:Timestamp>`: contains a time stamp defined in the *WS-Security* specification,
- 173 8. `<wsse:SecurityTokenReference>`: contains a reference to a security token, either using the
174 generic ID and reference mechanism, or a `<wsse:SecurityTokenReference>` element defined in
175 the *WS-Security* specification. A `<wsse:SecurityTokenReference>` element may contain a
176 `<wsse:KeyIdentifier>` element containing a key identifier type that is defined in *WS-Security*.

177 Each of these elements defines some XML attributes and sub-elements, but is also extensible. New
178 extension elements, such as new token types, etc. may also be added. Several token types are defined
179 in *WS-Security* profiles: *UsernameToken Profile* and *X.509 Token Profile*.

3 Policies about WS-Security

180

181 Before starting, it is important to understand the target of a “*WS-Security* policy”. In some cases, policy
182 writers want to constrain the content of instances of the “Security” headers defined by *WS-Security*. In
183 other cases, policy writers may want to specify how these “Security” headers are created and processed.
184 As an example of the first type of policy, the policy may specify acceptable values for the “Password”
185 element in a *WS-Security* “UserNameToken”. This type of policy constraint can be checked against the
186 actual content of a message containing such a “Password” element. As an example of the second type
187 of policy, the policy may specify that the type of password used in the “Password” element in a
188 “UserNameToken” should be a password digest rather than a plain-text password. This type of policy
189 constraint can’t be checked against the content of the message, since there is no element or attribute in
190 the “UserNameToken” that specifies the type of the password. The value of the “Password” element in
191 either case is a string. If the “Password” is incorrect, it is impossible in general to know whether it is
192 because the password value was incorrect or because it was supplied as a plain-text password rather
193 than as a password digest. If they are to communicate successfully, the producer of a “Security” header
194 containing a “Password” element must reach agreement with the consumers of that element about which
195 type of password will be used, but that agreement does not appear explicitly in the “Security” header
196 itself. Another example of policy information that does not appear in the message is a directive that new
197 “Security” headers should be pre-pended to existing ones. Producing messages in such a way aids
198 processing by message consumers, since “Security” headers often need to be verified in a particular
199 order (think of verifying a signature before decrypting the message versus decrypting the message first,
200 and then verifying the signature). But once again, the consumer of a message has no way of knowing
201 whether the producers of the “Security” headers actually pre-pended them: if the messages fail to verify,
202 the consumer has no way of knowing whether the headers were incorrect or whether they were not pre-
203 pended as expected.

204 Where policies concern information that does not appear in the “Security” header itself, a new policy
205 element or Attribute needs to be defined to express such information. Such elements or Attributes do
206 not need to be instantiated – they will never appear in a message – but they are needed as a way of
207 talking about how the “Security” header is to be created and consumed. For example, if the producer
208 and the consumer both agree that *PasswordType* = “*PasswordDigest*”, and they produce and consume
209 the message accordingly, then there is no need for *PasswordType* = “*PasswordDigest*” to appear in the
210 message itself. Nevertheless, it may be useful to convey explicitly in a message the fact that a
211 “*PasswordDigest*” is being used.

212 Where new elements or Attributes are found to be needed to specify policies about information not
213 currently specified in a message, it may be an indication that new elements should be added to the
214 underlying specification in a future revision. If such new elements are defined in the underlying
215 specification with default values, this will encourage consistent use of message contents without making
216 messages that conform to the default any longer than currently. Alternatively, the underlying
217 specification could mandate certain processing actions or behaviors to enable consistent usage of
218 instances of the specification's schema.

219

4 WS-SecurityPolicy

220 *WS-SecurityPolicy* defines the domain-specific policy Assertions to be used with *WS-Policy* when writing
221 policies about security information associated with a message. *WS-SecurityPolicy* is described as being
222 generic to any underlying security specification, and not confined to use with *WS-Security*. In practice,
223 *WS-SecurityPolicy* would have to be re-implemented for each underlying security specification if the
224 policies are to be verified, so most implementations can be expected to support only *WS-Security*.

225 When used with *WS-Security*, *WS-SecurityPolicy* defines 7 types of *WS-Policy* `<Assertion>` elements
226 that can be used to place constraints on the *WS-Security* `<wsse:Security>` header to be used in
227 SOAP messages:

- 228 1. `<SpecVersion>`: indicates support for *WS-Security*, including the Addendum.
- 229 2. `<SecurityToken>`: constrains the types and contents of security tokens supplied with a message.
- 230 3. `<Integrity>`: places constraints on digital signatures used in the message.
- 231 4. `<Confidentiality>`: places constraints on the use of encryption in the message.
- 232 5. `<Visibility>`: specifies portions of a message that must be able to be processed by an
233 intermediary or endpoint.
- 234 6. `<SecurityHeader>`: constrains aspects of the *WS-Security* `<wsse:Security>` header.
- 235 7. `<MessageAge>`: constrains the use of the `<wsse:Timestamp>` header from *WS-Security*

236 The `<Assertion>` elements defined in *WS-SecurityPolicy* include `wsp:Preference` and `wsp:Usage`
237 XML attributes. Since the most recent draft of *WS-Policy* omits these attributes, these are not described
238 below.

239 The schema for *WS-SecurityPolicy* [WSSP-Sch] is a collection of element definitions, many of which are
240 freely extensible. The `Assertion` elements in general are not structured in the schema itself, but
241 depend on using the extensibility of their parent elements. Putting the elements together in a meaningful
242 way requires studying the *WS-SecurityPolicy* specification.

243 Except for `<SpecVersion>`, `<SecurityHeader>`, and `<MessageAge>` the *WS-SecurityPolicy*
244 Assertions, are “generic”, and might be used to apply to any security parameter specification
245 mechanism. [Note: `<MessageAge>` could have been generic, but the *WS-SecurityPolicy* specification
246 specifically ties it to the *WS-Security* `<Timestamp>` element]. The *WS-SecurityPolicy* engine used to
247 process and enforce policies using these Assertions must include specific code modules to support the
248 application of the Assertions to each specification mechanism, however, so the engine must be modified
249 to support *WS-Security*. If the Assertions are used to apply to some other specification mechanism, the
250 *WS-SecurityPolicy* engine must be modified again to support the new specification. Just being “generic”
251 does not automatically make *WS-SecurityPolicy* work with any security parameter specification
252 mechanism.

5 Using WS-PolicyConstraints for WS-Security Policies

253

254

255

256 Many of the following *WS-PolicyConstraints* predicates are written directly against the *WS-Security*
257 specification. Where new elements or attributes are needed to specifying information that does not
258 appear in a message instance directly, the predicates are written against the element defined for this
259 purpose in the *WS-SecurityPolicy* specification. Note that using *WS-SecurityPolicy* in this way does not
260 mean that domain-specific policy processing modules are needed. The predicates are still expressed
261 using the generic *WS-PolicyConstraints* language and can be handled by a generic policy processor.
262 Using the element defined in *WS-SecurityPolicy* is only one possible approach. Using *WS-*
263 *PolicyConstraints* to express most policy constraints against the message itself means that additional
264 policy elements can usually be much simpler than those currently defined in *WS-SecurityPolicy*. In most
265 cases, a simple Attribute could be used instead.

266 Where predicates are written directly against the *WS-Security* specification, the predicates can be
267 directly enforced using an XACML Policy Decision Point engine, although the engine must be extended
268 to support some new functions and datatypes. It is expected that the number of such extensions will be
269 limited, since they are needed only for expressing policies about legacy data that is not in XML, such as
270 the contents of public key certificates. Future information appears likely to be defined in XML, and can
271 then be referenced using the existing standard XACML functions.

272 If security-related predicates are to be written using *WS-PolicyConstraints* against some schema other
273 than *WS-Security*, then the predicates would need to be reformulated. Since two parties in a message
274 exchange must agree on how they will specify their security information, however, it does not seem
275 overly restrictive to require that the actual policies be written against the specific format especially since
276 one of the payoffs is the ability to directly verify the policy against the messages. If more abstract
277 policies are needed, then new abstract elements can be defined (or re-use the ones defined in *WS-*
278 *SecurityPolicy*) and associated with sets of specific predicates for particular security header schemas.

5.1 Multiple constraints on a single nodeset

280 Frequently, a policy will require a single nodeset in a `<wsse:Security>` header to satisfy multiple
281 conditions. In these cases, the *WS-PolicyConstraints* `&wspc;function:limit-scope` may be used to
282 enclose the predicates that must be satisfied within a single nodeset. This function is defined as an
283 extension to XACML using XACML's extensible function capabilities. For example, if a particular
284 canonicalization method and a particular signature method must be used in a single `<ds:Signature>`
285 element, the following *WS-PolicyConstraints* predicate would be used.

```
286 <Apply FunctionId="&wspc;function:limit-scope">  
287   <AttributeValue  
288     DataType="&xsd:string">//S11:Envelope/S11:Header/wsse:Security/ds:Signat  
289     ure/ds:SignedInfo</AttributeValue>  
290     <Apply FunctionId="&xfunc:anyURI-equal">  
291       <AttributeSelector DataType="&xsd:string" RequestContextPath=  
292         "/ds:CanonicalizationMethod/@Algorithm".>  
293       <AttributeValue DataType="&xsd:anyURI">  
294         http://www.w3.org/2001/10/xml-exc-c14n"</AttributeValue>  
295     </Apply>  
296     <Apply FunctionId="&xfunc:anyURI-equal">  
297       <AttributeSelector DataType="&xsd:anyURI" RequestContextPath=  
298         "/ds:SignatureMethod/@Algorithm".>  
299       <AttributeValue DataType="&xsd:anyURI">  
300         http://www.w3.org/2000/09/xmldsig#hmac-sha1</AttributeValue>  
301     </Apply>  
302 </Apply>
```

303 Only individual predicates will be shown below, but it should be remembered that the
304 `&wspc;function:limit-scope` function may be used to restrict any set of predicates to a single
305 nodeset.

306 **5.2 Limited XPath expressions**

307 In order for policy predicates in two different policies to be compared, it is necessary to be able to tell
308 whether the two predicates refer to the same underlying policy vocabulary item. Using full XPath, there
309 are multiple ways to refer to the same element or attribute in schema instances, and it is not possible to
310 determine the same element or attribute is being referenced.

311 *WS-PolicyConstraints* needs to use a subset of XPath such that it can be determined whether any two
312 XPath expressions refer to the same nodeset or nodesets. No such subset has been defined as far as
313 we know.

314 A proposed subset is used in these examples. This subset uses only absolute XPath expressions (i.e.
315 all start with //<doc root>), does not use any numbered elements (e.g. x[1]), does not use query functions
316 in the XPath expressions (e.g. [@PasswordType="PasswordDigest"]), and references only text elements
317 or the values of XML attributes in the terminal element of the XPath expression.

318 This subset may be inadequate and is probably overly constrained, but further research is needed to
319 specify an optimal subset that retains as much expressivity as possible while preserving the ability to
320 match the potential nodesets specified by each XPath expression.

6 Actual WS-Security policy predicates

321

322 This section contains actual predicates specified using *WS-PolicyConstraints* for each predicate defined
323 in a *WS-SecurityPolicy* Assertion.

6.1 Specification version

324

325 *WS-SecurityPolicy* uses the `<wsp:SpecVersion>` Assertion to indicate 'support for *WS-Security*
326 including the Addendum". The example provided is:

```
327 <wsp:SpecVersion URI="http://schemas.xmlsoap.org/ws/2002/07/secext"/>
```

328 The semantics of this Assertion are not entirely clear: is it intended to specify that a particular version of
329 *WS-Security* must be used, or that there must be an instance of a *WS-Security* `<wsse:Security>`
330 header using this version of the specification?

331 *WS-PolicyConstraints* can express either or both of these policy requirements precisely as follows.

332 The following predicate requires that a particular version of *WS-Security* be used. This type of predicate
333 is called a "WS-Security version predicate" in the subsequent discussion.

```
334 <Apply FunctionId="&xfunc:anyURI-is-in">  
335 <AttributeValue DataType="&xsd:anyURI">  
336 http://schemas.xmlsoap.org/ws/2002/07/secext</AttributeValue>  
337 <AttributeSelector  
338 RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/@xmlns"  
339 DataType="&xsd:anyURI"/>  
340 </Apply>
```

341 If more than one version of *WS-Security* is supported, then multiple instances of a "WS-Security version
342 predicate" may be used as follows:

```
343 <wsp:ExactlyOne>  
344 <...WS-Security version predicate #1.../>  
345 <...WS-Security version predicate #2.../>  
346 </wsp:ExactlyOne>
```

347 In general, when using *WS-PolicyConstraints*, no predicate used merely to require the presence of a
348 `<wsse:Security>` header will be needed, since other predicates that require the header to contain
349 particular contents will occur and can only be satisfied if the header itself is present, as in the "*WS-*
350 *Security* version predicate" above. Nevertheless, if it is actually the case that some form of security
351 header must be present, but without any constraints on its contents, the following *WS-PolicyConstraints*
352 predicate can be used. This predicate is called "Security header present predicate" below.

```
353 <Apply FunctionId="&wspc;function:must-be-present">  
354 <AttributeValue  
355 DataType="&xsd:string"/>S11:Envelope/S11:Header/wsse:Security</Attribute  
356 Value>  
357 </Apply>
```

358 If the `<wsse:Security>` header is optional, then "Security header present predicates" may be used as
359 follows, where "other predicates" are other predicates that must be satisfied in addition to the "Security
360 header present predicate":

```
361 <wsp:ExactlyOne>  
362 <wsp>All>  
363 ...other predicates...  
364 <...Security header present predicate.../>  
365 </wsp>All>  
366 <wsp>All>  
367 ...other predicates...  
368 </wsp>All>  
369 </wsp:ExactlyOne>
```

370 Alternatively, the *WS-PolicyConstraints* function "`&wspc;function:must-not-be-present`" can be
371 used. This may be especially attractive if the number of other, independent predicates is large. The
372 policy above would then look as follows:

```

373 <wsp:ExactlyOne>
374 <wsp:All>
375   ...other predicates not depending on Security header...
376   <wsp:ExactlyOne>
377     <wsp:All>
378       <...Security header present predicate.../>
379       ...other predicates depending on Security header...
380     </wsp:All>
381     <Apply FunctionId="&wspc;function:must-not-be-present">
382       <AttributeValue
383   DataType="&xsd:string">//S11:Envelope/S11:Header/wsse:Security</Attribut
384   eValue>
385     </wsp:ExactlyOne>
386   </wsp:All>
387 </wsp:ExactlyOne>

```

388 6.2 Security tokens

389 *WS-SecurityPolicy* uses the `<wsp:SecurityToken>` Assertion to “describe what security tokens are
390 required and accepted by a Web service. It can also be used to express a Web Service's policy on
391 security tokens that are included when the service sends out a message (e.g., as a reply message).”

392 *WS-Security* does not define any standard elements or attributes for a security token: all security tokens
393 are defined in profiles. *WS-PolicyConstraints* examples are provided below for each token type listed in
394 Appendix I of *WS-SecurityPolicy*. If new token types are defined, the *WS-SecurityPolicy* specification
395 would need to be amended to support them, whereas corresponding *WS-PolicyConstraints* predicates
396 require only knowledge of the new security token profile schema, and can be written using the standard
397 *WS-PolicyConstraints* language.

398 There are a number of sub-elements included in a `<wsp:SecurityToken>` Assertion. We will first
399 describe those common to various security token profiles, then those specific to particular profiles, each
400 with its corresponding *WS-PolicyConstraints* predicate. Then we will show how the *WS-*
401 *PolicyConstraints* “limit-scope” function can be used to require that a collection of predicates must all be
402 true for a single security token in the `<wsse:Security>` header.

403 6.2.1 Common Token Profile Elements

404 `/wssp:SecurityToken/wssp:TokenType`

405 There are actually three token types defined in the *Web Services Security X.509 Certificate Token*
406 *Profile*. The *Profile* states that these URI fragments are relative to the URI for “this specification”, but
407 that URI is not clearly stated anywhere. The “Document Location” on the title page is shown as
408 “http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0”, so perhaps that is
409 what is meant. We use the XML Internal Entity “&x509;” to refer to the correct URI.

- 410 1. &x509;#X509v3: a single X.509 v3 signature-verification certificate.
- 411 2. &x509;#X509PKIPathv1: an ordered list of X.509 certificates packaged in a PKIPath.
- 412 3. &x509;#PKCS7: a list of X.509 certificates, and (optionally) CRLs packaged in a PKCS#7 wrapper.

413 *WS-PolicyConstraints* allows specification of each token type precisely:

```

414 <Apply FunctionId="&wspc;must-be-present">
415   <AttributeValue
416   DataType="&xsd:string">//S11:Envelope/S11:Header/wsse:Security/&x509;#x5
417   09v3</AttributeValue>
418 </Apply>

```

419 where the desired X.509 or other security token type would be specified in the XPath expression used in
420 the `<AttributeValue>`.

421 `/wssp:SecurityToken/wssp:TokenIssuer`

422 In an actual X.509 certificate, the Issuer will be embedded in the binary data of the certificate itself. If the

423 value in the certificate is required, then *WS-PolicyConstraints* will require that a new function must be
424 written to extract that value from a certificate reference. Note that any implementation of *WS-*
425 *SecurityPolicy* in which this Assertion will be applied to an actual X.509 certificate will require
426 implementation of an equivalent function internally.

427 If, rather than the value in an X.509 certificate, the policy refers to the value specified in an instance from
428 the *XML Digital Signature* standard that includes the issuer name, then no new function is required. The
429 following example shows a predicate that uses the *XML Digital Signature* element
430 <ds:X509IssuerName>, that must be present in the <ds:KeyInfo> in the <wsse:Security>
431 header. This example requires that the certificate issuer name be "DC=ACMECorp, DC=com".

```
432 <Apply FunctionId="&xfunc;x509Name-match">  
433   <Apply FunctionId="&xfunc;x509Name-one-and-only">  
434     <AttributeSelector  
435       RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:KeyInfo/w  
436       sse:SecurityTokenReference/ds:X509Data/ds:X509IssuerSerial/ds:X509Issuer  
437       Name/text()" DataType="&xdata;x509Name"/>  
438     </Apply>  
439     <AttributeValue DataType="&xdata;x509Name">DC=ACMECorp,  
440     DC=com</AttributeValue>  
441   </Apply>
```

442 Token issuer constraints can similarly be constructed for other types of security tokens.

443 **/wssp:SecurityToken/wssp:Claims/wssp:SubjectName**

444 As with the previous "TokenIssuer" Assertion element, a "SubjectName" Assertion might refer to an
445 actual X.509 certificate included in a message, or to a description of such a certificate specified using
446 *XML Digital Signature*. As before, if the policy refers to the contents of an actual X.509 certificate, then
447 *WS-PolicyConstraints* will require that a new function be written to extract that value from a certificate
448 reference. Any implementation of *WS-SecurityPolicy* in which this Assertion element will constrain an
449 actual X.509 certificate instance will require implementation of an equivalent function.

450 The following *WS-PolicyConstraints* predicate assumes that the subject name from the contents of the
451 X509 certificate is provided in the *XML Digital Signature* description contained in the Security header.
452 This example requires that the certificate subject's name be "cn=Anne.Anderson, DC=Sun, DC=com".

```
453 <Apply FunctionId="&xfunc;x509Name-match">  
454   <Apply FunctionId="x509Name-one-and-only">  
455     <AttributeSelector  
456       RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:KeyInfo/w  
457       sse:SecurityTokenReference/ds:X509Data/ds:X509SubjectName/text()"  
458       DataType="&xdata;x509Name"/>  
459     </Apply>  
460     <AttributeValue DataType="&xdata;x509Name">CN=Anne.Anderson, DC=Sun,  
461     DC=com</AttributeValue>  
462   </Apply>
```

463 Subject name constraints can similarly be constructed for other types of security tokens.

464 **/wssp:SecurityToken/wssp:Claims/wssp:SubjectName/@wssp:MatchType**

465 The preceding example required the subject name to match the supplied value exactly. *WS-*
466 *SecurityPolicy* provides an XML attribute to be used for specifying either an exact match or a match
467 where the specified value must be the prefix of the value in the certificate. In *WS-PolicyConstraints*, this
468 information is provided via the matching function that is used. For example, if a prefix match is desired,
469 then the following *WS-PolicyConstraints* predicate may be used. In this example, the subject's X500
470 name must begin with "CN=Anne.Anderson," where "CN" may be capitalized or not.

```
471 <Apply FunctionId="&xfunc;x500Name-regex-match">  
472   <AttributeValue DataType="&xsd:string">^[Cc][Nn] *=  
473   *Anne.Anderson.*</AttributeValue>  
474   <Apply FunctionId="x500Name-one-and-only">  
475     <AttributeSelector  
476       RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:KeyInfo/ws  
477       se:SecurityTokenReference/ds:X509Data/ds:X509SubjectName/text()"  
478       DataType="&xdata;x500Name"/>
```

479
480

```
</Apply>  
</Apply>
```

481 6.2.2 X.509v3 Token

482 /wssp:SecurityToken/wssp:Claims/wssp:X509Extension

483 This element in a *WS-SecurityPolicy* <SecurityToken> specifies the value, OID, and (optionally) the
484 criticality of a required X509Extension in the certificate. The *XML Digital Signature* specification does not
485 contain elements for describing extensions contained in an X509 certificate other than the
486 SubjectKeyInfo extension, so new functions must be written to extract this type of information from a
487 referenced certificate in a message.

488 The following example uses a new XACML extension function called “hexBinary-
489 getCertExtensionValue”. It takes as input a string value interpreted as the OID of the desired
490 extension, a string value indicating required criticality (with acceptable values of “Critical”,
491 “NotCritical”, and “CriticalOrNot”), and a reference to an ASN-encoded X.509 certificate,
492 encoded in ASN.1 with data type “&xml;hexBinary”. It returns a bag containing the values of all
493 extensions in the referenced certificate that match the requirements. The values are returned with data
494 type “&xml;hexBinary”.

495 The following example requires the value of the extension having OID “1.2.840.113549.1.1.5”,
496 without regard to criticality, to match the value “0xFFABC123” in hex binary form. In this case there may
497 be multiple certificates in the message, and by used of the “&xfunc;any-of” function, it is required that
498 at least one of them have this extension with this value.

```
499 <Apply FunctionId="&wspc;function:limit-scope">  
500   <AttributeValue DataType="&xsd:string">  
501     //S11:Envelope/S11:Header/wsse:Security/wsse:BinarySecurityToken  
502   </AttributeValue>  
503   <Apply FunctionId="&xfunc;anyURI-equal">  
504     <AttributeValue DataType="&xsd:anyURI">#X509v3</AttributeValue>  
505     <AttributeSelector DataType="&xsd:anyURI"  
506       RequestContextPath="/@ValueType"/>  
507   </Apply>  
508   <Apply FunctionId="&xfunc;anyURI-equal">  
509     <AttributeValue DataType="&xsd:anyURI">#hexBinary</AttributeValue>  
510     <AttributeSelector DataType="&xsd:anyURI"  
511       RequestContextPath="/@EncodingType"/>  
512   </Apply>  
513   <Apply FunctionId="&xfunc;any-of">  
514     <Apply FunctionId="&xfunc;hexBinary-match">  
515       <AttributeValue  
516         DataType="&xsd;hexBinary">FFABC123</AttributeValue>  
517       <Apply FunctionId="&wspc;function:hexBinary-  
518         getCertExtensionValue">  
519         <AttributeValue DataType="&xsd:string">  
520           1.2.840.113549.1.1.5</AttributeValue>  
521         <AttributeValue DataType="&xsd:string">  
522           CriticalOrNot</AttributeValue>  
523         <AttributeSelector RequestContextPath="/text()" />  
524         DataType="&xsd;hexBinary"/>  
525       </Apply>  
526     </Apply>  
527   </Apply>
```

528 6.2.3 Kerberos Token

529 6.2.3.1 /wssp:SecurityToken/wssp:Claims/wssp:ServiceName

530 This element in a *WS-SecurityPolicy* <SecurityToken> is used with a Kerberos token to specify the
531 service's PrincipalName (sname field defined in RFC1510). The draft *Web Services Security*
532 *Kerberos Token Profile 1.0* specifies that a Kerberos ticket is included in a <wsse:Security> header
533 using the <wsse:BinarySecurityToken> described in *WS-Security*. Since this token is not in an

534 XML format, a special function must be written to extract the `sname` field.

535 The following *WS-PolicyConstraints* version of this constraint uses a new XACML extension function
536 called "string-getKrb5SName". It takes as input a reference to a Kerberos ticket in a
537 `<wsse:BinarySecurityToken>`. It returns the service's `sname` as a string.

538 The following example requires the value of the service name to match the regular expression
539 `".*\..WORLD"`. In this case there may be Kerberos tickets in the message, and it is required that at least
540 one of them have this service name.

```
541 <Apply FunctionId="&wspc;function:limit-scope">
542   <AttributeValue DataType="&xsd:string">
543     //S11:Envelope/S11:Header/wsse:Security/wsse:BinarySecurityToken/
544   </AttributeValue>
545   <Apply FunctionId="&xfunc;anyURI-equal"> <!-- valueType -->
546     <AttributeValue DataType="&xsd:anyURI">
547       http://www.docs.oasis-open.org/wss/2004/07/oasis-000000-wss-kerberos-
548       token-profile-1.0#Kerberosv5_AP_REQ
549     </AttributeValue>
550     <AttributeSelector DataType="&xsd:anyURI" RequestContextPath=
551       "@ValueType"/>
552   </Apply>
553   <Apply FunctionId="&xfunc;anyURI-equal"> <!-- encodingType -->
554     <AttributeValue DataType="&xsd:anyURI">#base64Binary
555   </AttributeValue>
556     <AttributeSelector DataType="&xsd:anyURI" RequestContextPath=
557       "@EncodingType"/>
558   </Apply>
559   <Apply FunctionId="&xfunc;any-of">
560     <Apply FunctionId="&xfunc;string-regex-match">
561       <AttributeValue
562         DataType="&xsd:string">.*\..WORLD</AttributeValue>
563       <Apply FunctionId="&wspc;function:string-getKrb5SName">
564         <AttributeSelector RequestContextPath=
565           "/text()" DataType="&xsd;base64Binary"/>
566       </Apply>
567     </Apply>
568   </Apply>
569 </Apply>
```

570 6.2.4 Username Token

571 6.2.4.1 /wssp:SecurityToken/wssp:Claims/wssp:UsePassword

572 This element in *WS-SecurityPolicy* specifies the requirements on the `<wsse:Password>` element in the
573 `<wsse:UsernameToken>`. The requirements are included in a `Type` XML attribute that may have the
574 value `wsse:PasswordText` or `wsse:PasswordDigest`. The *Web Services Security Username*
575 *Token Profile 1.0* extends the `<UsernameToken>` element defined in *WS-Security* with a "Password"
576 element having a "Type" attribute.

577 Using *WS-PolicyConstraints*, requirements on the values for this "Type" attribute can be expressed as
578 follows.

```
579 <Apply FunctionId="&xfunc;anyURI-is-in">
580   <AttributeValue DataType="&xsd:anyURI">#PasswordText</AttributeValue>
581   <AttributeSelector
582     DataType="&xsd:anyURI"
583     RequestContextPath=
584     //S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:Passwor
585     d/@Type"/>
586 </Apply>
```

587 6.3 Integrity Assertion

588 *WS-SecurityPolicy* uses the `<wssp:Integrity>` Assertion to indicate required signature formats.

589 The following elements and XML attributes are used in this Assertion and can be expressed using the
590 *WS-PolicyConstraints* predicates described below.

591 **6.3.1 /wssp:Integrity/wssp:Algorithm/@wssp:Type**

592 This XML Attribute is used in a <wssp:Integrity> Assertion to indicate an algorithm type, where the
593 values can be wsse:AlgCanonicalization, wsse:AlgSignature, or wsse:AlgTransform.

594 In *WS-PolicyConstraints* the algorithm type will be specified by the path selected for the algorithm
595 identifier in the next example.

596 **6.3.2 /wssp:Integrity/wssp:Algorithm/@wssp:URI**

597 This XML Attribute is used in a <wssp:Integrity> Assertion to indicate the URI associated with an
598 algorithm. The following *WS-PolicyConstraints* predicate can be used to indicate that a canonicalization
599 algorithm with the URI "http://www.w3.org/2001/10/xml-exc-c14n" is required.

```
600 <Apply FunctionId="&xfunc;anyURI-is-in">  
601   <AttributeValue DataType="&xsd:anyURI">  
602     http://www.w3.org/2001/10/xml-exc-c14n</AttributeValue>  
603   <AttributeSelector DataType="&xsd:anyURI"  
604     RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:SignedInf  
605     o/ds:CanonicalizationMethod/@Algorithm"/>  
606 </Apply>
```

607 **6.3.3 /wssp:Integrity/wssp:TokenInfo/wssp:SecurityToken**

608 This element "indicates a supported security token format or authority previously described". This
609 element is under-specified in *WS-SecurityPolicy* so it is not possible to determine for sure what the
610 contents of of such a <wssp:SecurityToken> element should be. If the value is a
611 <wsse:Reference> to a previously specified <wssp:SecurityToken> element, then in *WS-*
612 *PolicyConstraints*, this Assertion would be replaced with a predicate that references the
613 <wsse:SecurityTokenReference> element in the signature's <ds:KeyInfo> element, as follows.

```
614 <Apply FunctionId="&xfunc;anyURI-is-in">  
615   <AttributeValue DataType="&xsd:anyURI">#X509Token</AttributeValue>  
616   <AttributeSelector DataType="&xsd:anyURI" RequestContextPath=  
617     "//S11:Envelope/S11:Header/wsse:Security/s:Signature/ds:KeyInfo/wsse:Sec  
618     urityTokenReference/wsse:Reference/@URI"/>  
619 </Apply>
```

620 **6.3.4 /wssp:Integrity/wssp:Claims**

621 This element "contains data that is interpreted as describing general claims that must be expressed in
622 the security token." In *WS-SecurityPolicy*, the specific claims would have to be specified in some
623 extension document, along with their interpretation, matching, and verification semantics.

624 In *WS-PolicyConstraints*, claims are just constraints. These can be created without requiring any
625 extensions. For example, if some particular string value is required in a UsernameToken, this could be
626 expressed as follows:

```
627 <Apply FunctionId="&xfunc;string-is-in">  
628   <AttributeValue DataType="&xsd:string">...required  
629   value...</AttributeValue>  
630   <AttributeSelector DataType="&xsd:string"  
631     RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/wsse:Usernam  
632     eToken/...path to location of required value..."/>  
633 </Apply>
```

634 For stating constraints about contents of BinarySecurityToken types, new XACML extension functions
635 will usually be needed to extract the desired field from the encoded token. Any implementation of *WS-*
636 *SecurityPolicy* must also have such extraction functions implemented internally.

637 **6.3.5 /wssp:Integrity/wssp:MessageParts**

638 This element's text contents "is an expression that specifies the targets to be signed. The evaluation of
639 the expression is determined by the optional dialect attribute."

640 In *WS-PolicyConstraints*, this type of Assertion would be replaced with a predicate that constrains the
641 contents of the `ds:Reference/@URI` XML attribute in the `<ds:SignedInfo>` element. For example,
642 a requirement that the “Body” of the message must be signed would be expressed as follows:

```
643 <Apply FunctionId="&xfunc:anyURI-is-in">  
644   <AttributeValue DataType="&xsd:anyURI">#body</AttributeValue>  
645   <AttributeSelector DataType="&xsd:anyURI"  
646   RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:Signature  
647   /ds:SignedInfo/ds:Reference/@URI"/>  
648 </Apply>
```

649 No specific predicate is needed in *WS-PolicyConstraints* to express the `<wssp:MessageParts`
650 `[@dialect]>` XML attribute, since *WS-PolicyConstraints* always uses XPath in its
651 `<AttributeSelector>` elements.

652 **6.3.6 /wssp:Integrity/wssp:MessageParts/@Signer**

653 In *WS-SecurityPolicy*, “this optional attribute contains a list of one or more URI references that indicate
654 which nodes must provide a signature”, where the pre-defined value is the *WS-Security* URI associated
655 with the role or actor XML attribute in the `<wsse:Security>` header element.

656 In *WS-PolicyConstraints*, this component of the `<wssp:Integrity>` Assertion can be expressed as
657 follows.

```
658 <Apply FunctionId="&wspc;function:limit-scope">  
659   <AttributeValue DataType="&xsd:string">  
660     //S11:Envelope/S11:Header/wsse:Security/</AttributeValue>  
661   <Apply FunctionId="&xfunc:anyURI-is-in">  
662     <AttributeValue DataType="&xsd:anyURI">  
663       ...Actor URI...</AttributeValue>  
664     <AttributeSelector RequestContextPath=  
665     "//S11:Envelope/S11:Header/wsse:Security/@S11:actor"/>  
666   </Apply>  
667   <Apply FunctionId="&wspc;function:must-be-present">  
668     <AttributeValue  
669     DataType="&xsd:string">/ds:Signature</AttributeValue>  
670   </Apply>  
671 </Apply>
```

672 This predicate would usually be used as part of one or more of the other predicates placing constraints
673 on the `<ds:Signature>` element for the specified actor.

674 **6.4 Confidentiality Assertion**

675 The `<wssp:Confidentiality>` Assertion places constraints on the use of encryption within the
676 message.

677 In *WS-PolicyConstraints*, the components of this Assertion would be replaced with predicates identical to
678 those specified above for the `<wssp:Integrity>` Assertion, except that the `<AttributeSelector>`
679 would select elements or XML attributes in the `//S11:Envelope/S11:Body/xenc:EncryptedData`
680 node. These predicates are not spelled out here as they are directly comparable to those shown above.

681 **6.5 Visibility Assertion**

682 The `<wssp:Visibility>` Assertion describes parts of the message that must either be unencrypted,
683 or must be encrypted for a particular `wsse:actor` or `wsse:role`.

684 In *WS-PolicyConstraints*, a requirement that parts of the message NOT be encrypted would be
685 expressed in a predicate such as the following, which requires that nothing in the body of the message
686 be encrypted.

```
687 <Apply FunctionId="&wspc;function:must-not-be-present">  
688   <AttributeSelector  
689   RequestContextPath="//S11:Envelope/S11:Body/xenc:EncryptedData"/>  
690 </Apply>
```

691 The *WS-SecurityPolicy* specification does not describe how a `<wssp:Visibility>` element specifies

692 an encryption target for a particular actor. The example claims to require that the body be visible to the
693 "http://www.fabrikan123.com" endpoint, but that detail seems to have been omitted.

694 A requirement that a particular part of the message be encrypted for a particular intermediary would be
695 expressed in a predicate such as the following, which requires that the message contain an encrypted
696 key intended for recipient "http://www.fabrikan123.com".

```
697 <Apply FunctionId="&xfunc;string-is-in"/>  
698 <AttributeValue DataType="&xsd;string">  
699 http://www.fabrikan123.com</AttributeValue>  
700 <AttributeSelector DataType="&xsd;string" RequestContextPath=  
701 //S11:Envelope/S11:Header/enc:EncryptedData/ds:KeyInfo/enc:EncryptedKey  
702 /@Recipient/>  
703 </Apply>
```

704 This could be combined with a constraint indicating that this key is used to encrypt the <S11:Body> of
705 the message.

```
706 <Apply FunctionId="&wspc;function:limit-scope">  
707 <AttributeValue DataType="&xsd;string">  
708 //S11:Envelope/S11:Header/enc:EncryptedData</AttributeValue>  
709 <Apply FunctionId="&xfunc;string-is-in">  
710 <AttributeValue DataType="&xsd;string">  
711 http://www.fabrikan123.com</AttributeValue>  
712 <AttributeSelector DataType="&xsd;string" RequestContextPath=  
713 //ds:KeyInfo/enc:EncryptedKey/@Recipient"/>  
714 </Apply>  
715 <Apply FunctionId="&xfunc;anyURI-is-in">  
716 <AttributeValue DataType="&xsd;anyURI">#Body</AttributeValue>  
717 <AttributeSelector Datatype="&xsd;anyURI" RequestContextPath=  
718 "/enc:CipherData/enc:CipherReference/@URI"/>  
719 </Apply>  
720 </Apply>
```

721 6.6 Security Header Assertion

722 *WS-SecurityPolicy* uses the <wssp:SecurityHeader> Assertion to indicate requirements on the
723 <wsse:Security> elements in the header in a SOAP message.

724 The following elements and XML attributes are used in this Assertion and can be expressed using the
725 *WS-PolicyConstraints* predicates described below.

726 6.6.1 /SecurityHeader/@MustPrepend

727 This XML attribute is used to specify that, when new <wsse:Security> elements are added to a
728 SOAP message header, they must be prepended. This is helpful when processing order is important,
729 such as whether encryption is being done before or after signature generation.

730 This constraint is a requirement on the creation process for a *WS-Security* header rather than a
731 requirement on its content. In order to express this policy, a new policy variable or vocabulary item must
732 be created about which policy can be stated. As an example, we will use the new vocabulary item
733 defined in *WS-SecurityPolicy* for this, but this item could as easily be a new XACML Attribute or an XML
734 element simpler than the one defined in *WS-SecurityPolicy*. Note that, even though an element from
735 *WS-SecurityPolicy* is used in stating the policy, the policy processor does not need to understand
736 anything about *WS-SecurityPolicy*, and needs no new code in order to process this constraint.

```
737 <Apply FunctionId="&xfunc;boolean-is-in">  
738 <AttributeValue DataType="&xsd;boolean">true</AttributeValue>  
739 <AttributeSelector DataType="&xsd;boolean" RequestContextPath=  
740 //wsp:SecurityHeader/@MustPrepend"/>  
741 </Apply>
```

742 This requirement can not be enforced by a policy processor, as the receiver has no way of knowing
743 whether the operations were actually done in the order indicated by the header elements. One symptom
744 of failure to perform operations in the specified order is that signatures will fail to verify and encryption
745 cannot be successfully decrypted, but unless the receiver tries all alternative orderings, there is no way
746 to distinguish this case from the case in which an invalid signature or encryption has been done. Even

747 though the requirement can not be enforced, it is important for the sender and the receiver to be able to
748 agree on their policy.

749 Since header order can be important, it may be appropriate for a future revision or profile of *WS-Security*
750 specify that `<wsse:Security>` header elements must occur in the order in which they are to be
751 processed.

752 **6.6.2 /SecurityHeader/@MustManifestEncryption**

753 This XML attribute “indicates that only encryptions listed or referenced from the `<Security>` header will
754 be processed; any encryptions in the message not referenced will be ignored. If false (the default), then
755 the processor MUST search the message for applicable encryptions to process.”

756 As with “@MustPrepend” above, this is a requirement on the creation and processing of a SOAP
757 message and its `<wsse:Security>` headers rather than a requirement on the content of the message
758 itself. In order to express this policy, a new policy variable or vocabulary item must be created about
759 which policy can be stated. As an example, we will use the new vocabulary item defined in *WS-*
760 *SecurityPolicy* for this, but this item could as easily be a new XACML Attribute or an XML element
761 simpler than the one defined in *WS-SecurityPolicy*. Note that, even though an element from *WS-*
762 *SecurityPolicy* is used in stating the policy, the policy processor does not need to understand anything
763 about *WS-SecurityPolicy*, and needs no new code in order to process this constraint.

```
764 <Apply FunctionId="&xfunc:boolean-is-in">  
765 <AttributeValue DataType="&xsd:boolean">true</AttributeValue>  
766 <AttributeSelector DataType="&xsd:boolean" RequestContextPath=  
767 "///wsp:SecurityHeader/@MustManifestEncryption"/>  
768 </Apply>
```

769 This requirement can not be enforced by a policy processor, as the receiver has no way of knowing
770 whether all encrypted elements in the body that the sender intended to have processed were actually
771 referenced from the `<wsse:Security>` header. Even though the requirement can not be enforced, it is
772 important for the sender and the receiver to be able to agree on their policy.

773 **6.7 MessageAge Assertion**

774 *WS-SecurityPolicy* uses the `<wssp:MessageAge>` Assertion to indicate requirements on the
775 `<wsse:TimeStamp>` element in the `<Security>` Header in a SOAP message.

776 The following elements and XML attributes are used in this Assertion and can be expressed using the
777 *WS-PolicyConstraints* predicates described below.

778 **6.7.1 /MessageAge/@Age**

779 This XML attribute “specifies the actual maximum age timeout for a message expressed in seconds.”

780 In *WS-PolicyConstraints*, such requirements can be specified by placing constraints on the value of the
781 `<wsse:TimeStamp>` element itself. For example, the following constraint requires a maximum age of
782 3600 seconds. We use the existing XACML Attribute for “current time”.

```
783 <Apply FunctionId="&xfunc:dateTime-greater-than-or-equal">  
784 <Apply FunctionId="&xfunc:dateTime-add-dayTimeDuration">  
785 <Apply FunctionId="&xfunc:dateTime-one-and-only">  
786 <AttributeSelector  
787 RequestContextPath="/S11:Envelope/S11:Header/wsse:Security/wsu:Timestamp  
788 /wsu:Created/text()" DataType="&xsd:dateTime"/>  
789 </Apply>  
790 <AttributeValue  
791 DataType="&xqo;dayTimeDuration">3600S</AttributeValue>  
792 </Apply>  
793 <xacml:EnvironmentAttributeDesignator  
794 AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"  
795 DataType="&xsd:dateTime"/>  
796 </Apply>
```

797

7 Lessons learned and future work

798 This exercise in translating the `Assertions` defined in *WS-SecurityPolicy* for use with *WS-Security* has
799 yielded some new perspectives on the problem of expressing policies, and has also reinforced some
800 others.

801 7.1 XPath intersections

802 Most importantly, we need a subset of *XPath* to use in referring to nodes in instances of a schema, such
803 as the schema for *WS-Security*, such that it is possible to detect when two different policies refer to the
804 same node. Use of *XPath* expressions that do not use functions and or special node selectors such as
805 “`<path>[2]`” seems to be adequate for the requirements of expressing *WS-SecurityPolicy* constraints,
806 but more verification is needed.

807 7.2 New functions

808 The following new functions were found to be needed and have been added to Draft 5 of the *WS-*
809 *PolicyConstraints* specification.

- 810 1) `&wspc;function:string-to-uri`: converts a value of type `string` (that can be interpreted as
811 a valid URI) to a value of type `anyURI`. XACML has a `url-string-concatenate` function, but
812 this is not capable of composing a URI from fragments that are themselves not valid URIs (`#body`).
813 More flexible composition of URIs may be needed for matching references in the body of the SOAP
814 message to the corresponding elements in the SOAP header. This draft managed to express all
815 requirements from *WS-PolicyConstraints* without this function, but it may be needed for more specific
816 correlation requirements.
- 817 2) “`&wspc;hexBinary-getCertExtensionValue`”. It takes as input a string value interpreted as the
818 OID of the desired extension, a string value indicating required criticality (with acceptable values of
819 “`Critical`”, “`NotCritical`”, and “`CriticalOrNot`”), and a reference to an ASN-encoded X.509
820 certificate, encoded as `&xsd;hexBinary`. A corresponding function that takes a certificate encoded
821 in `&xsd;base64Binary` may also be needed. Functions that return the extension value as
822 `&xsd;string` or `&xsd;integer` may also be useful.
- 823 3) “`&wspc;string-getKrb5SName`”. It takes as input a reference to a Kerberos ticket in a
824 `<wsse:BinarySecurityToken>` that is encoded in `&xsd;base64Binary`. It returns the
825 service's `sname` as a string. Two versions of this function may actually be needed, one for the case
826 where the ticket is encoded in `&xsd;base64Binary` and the other where the ticket is encoded in
827 `&xsd;hexBinary`.

828 Note that all functions other than the first are for dealing with non-XML data embedded in the *WS-*
829 *Security* instance. Implementations of *WS-SecurityPolicy* must supply similar functions internally in order
830 to deal with these types of `Assertions`.

831 If the solution to the problem of matching a URI reference and target is to create a `&wspc;string-`
832 `url-concatenate` function, then *WS-PolicyConstraints* must support embedding this function in a
833 constraint.

834 7.3 Constraints on the message processor

835 Using only direct references to instances of *WS-Security*, it is not possible to create constraints that
836 impose requirements on the creation or processing of security aspects of a SOAP message. An
837 example is a requirement that `<Security>` elements in a header must be prepended, such that the
838 order in which they occur in the document is the reverse of the order in which the corresponding
839 operations were performed. This is an example of where some new policy element, such as the one
840 defined in *WS-SecurityPolicy* must be defined. In this draft of this profile, we have used the elements
841 defined in *WS-SecurityPolicy*, but new XACML Attributes or new simple XML elements could be used
842 instead. Since much of the *WS-SecurityPolicy* schema is vocabulary items that can actually be
843 referenced directly in the SOAP message, the *WS-SecurityPolicy* schema could be considerably

844 simplified and reduced if *WS-PolicyConstraints* were used. Also, if *WS-PolicyConstraints* is used, no
845 special processor will be needed for the *WS-SecurityPolicy* (or XACML) vocabulary items because they
846 are used simply as vocabulary item identifiers, and have no special semantics that the policy processor
847 must understand.

848 Since there is no way to verify this requirement in general, and since the typical use of the requirement is
849 to ensure correct processing of messages, it seems more reasonable to require use of a *WS-Security*
850 profile (or future revision of *WS-Security* itself) that always requires that `<Security>` elements in the
851 header be prepended.

852 This is a constraint on the message processor, and could also be handled by the creation of a schema
853 for describing characteristics of the message processor, with an instance of this schema included in the
854 *SOAP* `<S11:Header>`. Then *WS-PolicyConstraints* could be used to specify constraints directly on
855 such instances.

856 **7.4 Overly constrained policies**

857 In several cases, *WS-PolicyConstraints* requires that a *WS-Security* instance be constructed in a
858 particular way, when alternative expressions would be possible and equally valid. Multiple *WS-*
859 *PolicyConstraints* constraints could be OR'ed together to capture all such allowable alternatives, but it
860 might be simpler to define a profile of *WS-Security* that specifies one way of expressing a requirement
861 where multiple alternatives exist. Such a profile could be defined for use with *WS-Security* instances that
862 are to be used with *WS-Policy*. An example is various ways in which signature or encryption information
863 in the `<wsse:Security>` header might be referenced from the body of the SOAP message.

864 **7.5 Cost and value of abstraction**

865 *WS-SecurityPolicy* defines abstract ways to specify constraints. These abstract specifications are in
866 many cases easier to compare than the more specific constraints expressed in *WS-PolicyConstraints*.
867 Two considerations place this difference in a different perspective, however.

868 The first is that each abstraction in *WS-SecurityPolicy* must be resolved into specific forms in order to
869 enforce a given policy. In order to determine that a given instance of *WS-Security* conforms to a *WS-*
870 *SecurityPolicy* policy, the engine that implements *WS-SecurityPolicy* must be able to recognize all the
871 specific ways of expressing *WS-Security* and all the variations that are equally valid. A *WS-*
872 *PolicyConstraints* policy, however, can be used not only to match policies, but can also be used directly
873 to enforce the policy against a particular instance. It would probably be a good idea for the tools used to
874 create policies to support more abstract terms. The tool should then translate these terms into the
875 specific *WS-PolicyConstraints* predicates actually required to negotiate and verify policy.

876 The second is that, while it may be simpler to express an abstract constraint using *WS-SecurityPolicy*,
877 the processing of these abstract constraints is still complex: in fact, new processor code must be created
878 to handle every possible abstract constraint as applied to every possible concrete message format. With
879 a standard predicate language such as *WS-PolicyConstraints*, every possible constraint can be handled
880 by one standard processor. The simplicity should be provided at the policy authoring level, and not at
881 the concrete policy instance level.

8 References

882

- 883 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
884 RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.
- 885 **[SOAP]** M. Gudgin, et al., eds., *SOAP Version 1.2 Part 1: Messaging Framework*, W3C
886 Recommendation, 24 June 2003, [http://www.w3.org/TR/2003/REC-soap12-](http://www.w3.org/TR/2003/REC-soap12-part1-20030624/)
887 part1-20030624/.
- 888 **[WSP]** S. Bajaj, et al., *Web Services Policy Framework (WS-Policy)*, September
889 2004, <http://www.ibm.com/developerworks/library/specification/ws-polfram/>
- 890 **[WSPA]** T. Nadalin, ed., *WS-Policy Assertions*, 28 May 2003,
891 <http://www.ibm.com/developerworks/library/ws-polas>
- 892 **[WSPC]** A. Anderson, *XACML-Based Web Service Policy Constraint Language (WS-*
893 *PolicyConstraints)*, Working Draft 05, 27 June 2005,
894 <http://research.sun.com/projects/xacml/ws-policy-constraints-wd-05.pdf>.
- 895 **[WSPL]** T. Moses, ed., *XACML profile for Web-services*, OASIS Access Control
896 (XACML) TC, Working Draft 04, 29 September 2004, [http://www.oasis-](http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf)
897 [open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf](http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf).
- 898 **[WSR]** K. Iwasa, et al., eds., *WS-Reliability v1.1*, OASIS Web Service Reliable
899 Messaging TC, OASIS Standard, 15 November 2004, [http://www.oasis-](http://www.oasis-open.org/committees/download.php/9330/WS-Reliability-CD1.086.zip)
900 [open.org/committees/download.php/9330/WS-Reliability-CD1.086.zip](http://www.oasis-open.org/committees/download.php/9330/WS-Reliability-CD1.086.zip).
- 901 **[WSRP]** Stefan Batres, ed., *Web Services Reliable Messaging Policy Assertion (WS-RM*
902 *Policy)*, February 2005, [http://specs.xmlsoap.org/ws/2005/02/rm/WS-](http://specs.xmlsoap.org/ws/2005/02/rm/WS-RMPolicy.pdf)
903 RMPolicy.pdf
- 904 **[WSS]** T. Nadalin, et al., eds., *Web Services Security: SOAP Message Security 1.0*
905 *(WS-Security 2004)*, OASIS Web Services Security TC, OASIS Standard
906 200401, March 2004, [http://docs.oasis-open.org/wss/2004/01/oasis-200401-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
907 wss-soap-message-security-1.0.pdf.
- 908 **[WSS-Sch]** T. Nadalin, et al. eds., *WS-Security schema*, OASIS Web Services Security TC,
909 OASIS Standard, March 2004, [http://www.oasis-](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss)
910 open.org/committees/tc_home.php?wg_abbrev=wss.
- 911 **[WSSP]** T. Nadalin, ed., *Web Services Security Policy Language (WS-SecurityPolicy)*,
912 Version 1.0, 18 December 2002, [http://www.verisign.com/wss/WS-](http://www.verisign.com/wss/WS-SecurityPolicy.pdf)
913 SecurityPolicy.pdf
- 914 **[XACML]** T. Moses, ed., *eXtensible Access Control Markup Language (XACML) Version*
915 *2.0*, OASIS Access Control (XACML) TC, OASIS Standard, 1 February 2005,
916 [http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf)
917 os.pdf.
- 918 **[XDS]** Mark Bartel, et al., eds., *XML-Signature Syntax and Processing*, W3C
919 Recommendation, 12 February 2002, [http://www.w3.org/TR/2002/REC-xmlsig-](http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/)
920 core-20020212/
- 921 **[XENC]** Donald Eastlake, et al., eds., *XML Encryption Syntax and Processing*, W3C
922 Recommendation, 10 December 2002, <http://www.w3.org/TR/xmlenc-core/>.
- 923

A. Revision History

Rev	Date	By Whom	What
01	4 April 2005	Anne Anderson	Initial version: Replacing WS-SecurityPolicy with WS-PolicyConstraints. File name: Example-WS-SecurityPolicy
02	30 May 2005	Anne Anderson	Added much more introductory material.
03	28 June 2005	Anne Anderson	Changed title: WS-Security profile of WS-PolicyConstraints. Made examples consistent with "lessons learned" from previous versions. File name: ws-security-profile-of-ws-policy-constraints

926 **B. Notices**

927 Copyright © 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A.
928 All rights reserved.

929 Sun, Sun Microsystems, the Sun logo and Java are trademarks or registered trademarks of Sun
930 Microsystems, Inc. in the U.S. and other countries.

931 THIS DOCUMENT IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS,
932 REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF
933 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE
934 DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY
935 INVALID.