



1

2 **Web Services Security**

3 **UsernameToken Profile 1.1**

4 **OASIS Public Review Draft - 28 June 2005**

5 **OASIS identifier:**

6 {product-productVersion-artifactType-stage-descriptiveName-revision.form (Word) (PDF)
7 (HTML)}

8 **Location:**

9 <http://docs.oasis-open.org/wss/2005/xx/wss-v1.1-spec-pr-UsernameTokenProfile-01>

10 **Technical Committee:**

11 Web Service Security (WSS)

12 **Chairs:**

13 Kelvin Lawrence, IBM

14 Chris Kaler, Microsoft

15 **Editors:**

16 Anthony Nadalin, IBM

17 Chris Kaler, Microsoft

18 Ronald Monzillo, Sun

19 Phillip Hallam-Baker, Verisign

20 **Abstract:**

21 This document describes how to use the UsernameToken with the Web Services
22 Security (WSS) specification.

23 **Status:**

24 This is a technical committee document submitted for consideration by the OASIS Web
25 Services Security (WSS) technical committee. Please send comments to the editors.

26 If you are on the wss@lists.oasis-open.org list for committee members, send comments
27 there. If you are not on that list, subscribe to the wss-comment@lists.oasis-open.org list
28 and send comments there. To subscribe, send an email message to [wss-comment-
29 request@lists.oasis-open.org](mailto:wss-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

30 For patent disclosure information that may be essential to the implementation of this
31 specification, and any offers of licensing terms, refer to the Intellectual Property Rights
32 section of the OASIS Web Services Security Technical Committee (WSS TC) web page
33 at <http://www.oasis-open.org/committees/wss/ipr.php>. General OASIS IPR information
34 can be found at <http://www.oasis-open.org/who/intellectualproperty.shtml>.

35 Notices

36 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
37 that might be claimed to pertain to the implementation or use of the technology described in this
38 document or the extent to which any license under such rights might or might not be available;
39 neither does it represent that it has made any effort to identify any such rights. Information on
40 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
41 website. Copies of claims of rights made available for publication and any assurances of licenses
42 to be made available, or the result of an attempt made to obtain a general license or permission
43 for the use of such proprietary rights by implementors or users of this specification, can be
44 obtained from the OASIS Executive Director.

45 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
46 applications, or other proprietary rights which may cover technology that may be required to
47 implement this specification. Please address the information to the OASIS Executive Director.

48 Copyright © The Organization for the Advancement of Structured Information Standards [OASIS]
49 2002-2005. All Rights Reserved.

50 This document and translations of it may be copied and furnished to others, and derivative works
51 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
52 published and distributed, in whole or in part, without restriction of any kind, provided that the
53 above copyright notice and this paragraph are included on all such copies and derivative works.
54 However, this document itself does not be modified in any way, such as by removing the
55 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
56 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
57 Property Rights document must be followed, or as required to translate it into languages other
58 than English.

59 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
60 successors or assigns.

61 This document and the information contained herein is provided on an "AS IS" basis and OASIS
62 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
63 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
64 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
65 PARTICULAR PURPOSE.

66 **Table of Contents**

67 1 Introduction 4
68 2 Notations and Terminology..... 4
69 2.1 Notational Conventions 4
70 2.2 Namespaces 4
71 2.3 Acronyms and Abbreviations 5
72 3 UsernameToken Extensions 6
73 3.1 Usernames and Passwords 6
74 3.2 Token Reference..... 10
75 3.3 Error Codes 10
76 4 Key Derivation 10
77 5 Security Considerations..... 12
78 6 References 13
79 Appendix A. Acknowledgements 15
80 Appendix B. Revision History 17
81

82 1 Introduction

83 This document describes how to use the UsernameToken with the WSS: SOAP Message
84 Security specification [WSS]. More specifically, it describes how a web service consumer can
85 supply a UsernameToken as a means of identifying the requestor by "username", and optionally
86 using a password (or shared secret, or password equivalent) to authenticate that identity to the
87 web service producer.

88

89 This section is non-normative. Note that Sections 2.1, 2.2, all of 3, 4 and indicated parts of 6 are
90 normative. All other sections are non-normative.

91 2 Notations and Terminology

92 This section specifies the notations, namespaces, and terminology used in this specification.

93 2.1 Notational Conventions

94 The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
95 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be
96 interpreted as described in [RFC 2119].

97

98 When describing abstract data models, this specification uses the notational convention used by
99 the XML Infoset. Specifically, abstract property names always appear in square brackets (e.g.,
100 [some property]).

101

102 When describing concrete XML schemas [XML-Schema], this specification uses the notational
103 convention of WSS: SOAP Message Security. Specifically, each member of an element's
104 [children] or [attributes] property is described using an XPath-like [XPath] notation (e.g.,
105 /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element
106 wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard
107 (<xs:anyAttribute/>).

108

109 Commonly used security terms are defined in the Internet Security Glossary [SECGLO]. Readers
110 are presumed to be familiar with the terms in this glossary as well as the definition in the Web
111 Services Security specification.

112 2.2 Namespaces

113 Namespace URIs (of the general form "some-URI") represents some application-dependent or
114 context-dependent URI as defined in RFC 3986 [URI]. This specification is designed to work with
115 the general SOAP [SOAP11, SOAP12] message structure and message processing model, and
116 should be applicable to any version of SOAP. The current SOAP 1.1 namespace URI is used
117 herein to provide detailed examples, but there is no intention to limit the applicability of this
118 specification to a single version of SOAP.

119

120 The namespaces used in this document are shown in the following table (note that for brevity, the
 121 examples use the prefixes listed below but do not include the URIs – those listed below are
 122 assumed).
 123

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsse11	http://docs.oasis-open.org/wss/2005/xx/oasis-2005xx-wss-wssecurity-secext-1.1.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

124
 125 The URLs provided for the *wsse* and *wsu* namespaces can be used to obtain the schema files.
 126 URI fragments defined in this specification are relative to a base URI of the following unless
 127 otherwise stated:
 128 [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0)
 129 [profile-1.0](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0)
 130

131 The following table lists the full URI for each URI fragment referred to in this specification.
 132

URI Fragment	Full URI
#PasswordDigest	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest
#PasswordText	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText
#UsernameToken	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0 #UsernameToken

133 2.3 Acronyms and Abbreviations

134 The following (non-normative) table defines acronyms and abbreviations for this document.
 135

Term	Definition
SHA	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier

XML	Extensible Markup Language
-----	----------------------------

136 3 UsernameToken Extensions

137 3.1 Usernames and Passwords

138 The `<wsse:UsernameToken>` element is introduced in the WSS: SOAP Message Security
139 documents as a way of providing a username.

140

141 Within `<wsse:UsernameToken>` element, a `<wsse:Password>` element may be specified.
142 Passwords of type `PasswordText` and `PasswordDigest` are not limited to actual
143 passwords, although this is a common case. Any password equivalent such as a derived
144 password or S/KEY (one time password) can be used. Having a type of `PasswordText` merely
145 implies that the information held in the password is "in the clear", as opposed to holding a "digest"
146 of the information. For example, if a server does not have access to the clear text of a password
147 but does have the hash, then the hash is considered a *password equivalent* and can be used
148 anywhere where a "password" is indicated in this specification. It is not the intention of this
149 specification to require that all implementations have access to clear text passwords.

150

151 Passwords of type `PasswordDigest` are defined as being the Base64 [XML-Schema] encoded,
152 SHA-1 hash value, of the UTF8 encoded password (or equivalent). However, unless this digested
153 password is sent on a secured channel or the token is encrypted, the digest offers no real
154 additional security over use of `wsse:PasswordText`.

155

156 Two optional elements are introduced in the `<wsse:UsernameToken>` element to provide a
157 countermeasure for replay attacks: `<wsse:Nonce>` and `<wsu:Created>`. A nonce is a
158 random value that the sender creates to include in each `UsernameToken` that it sends. Although
159 using a nonce is an effective countermeasure against replay attacks, it requires a server to
160 maintain a cache of used nonces, consuming server resources. Combining a nonce with a
161 creation timestamp has the advantage of allowing a server to limit the cache of nonces to a
162 "freshness" time period, establishing an upper bound on resource requirements. If either or both
163 of `<wsse:Nonce>` and `<wsu:Created>` are present they **MUST** be included in the digest value
164 as follows:

165

166 Password_Digest = Base64 (SHA-1 (nonce + created + password))

167

168 That is, concatenate the nonce, creation timestamp, and the password (or shared secret or
169 password equivalent), digest the combination using the SHA-1 hash algorithm, then include the
170 Base64 encoding of that result as the password (digest). This helps obscure the password and
171 offers a basis for preventing replay attacks. For web service producers to effectively thwart replay
172 attacks, three counter measures are RECOMMENDED:

173

- 174 1. It is RECOMMENDED that web service producers reject any `UsernameToken` *not*
175 using *both* nonce *and* creation timestamps.

- 176 2. It is RECOMMENDED that web service producers provide a timestamp “freshness”
177 limitation, and that any UsernameToken with “stale” timestamps be rejected. As a
178 guideline, a value of five minutes can be used as a minimum to detect, and thus
179 reject, replays.
- 180 3. It is RECOMMENDED that used nonces be cached for a period at least as long as
181 the timestamp freshness limitation period, above, and that UsernameToken with
182 nonces that have already been used (and are thus in the cache) be rejected.

183

184 Note that the nonce is hashed using the octet sequence of its decoded value while the timestamp
185 is hashed using the octet sequence of its UTF8 encoding as specified in the contents of the
186 element.

187

188 Note that PasswordDigest can only be used if the plain text password (or password
189 equivalent) is available to both the requestor and the recipient.

190

191 Note that the secret is put at the end of the input and not the front. This is because the output of
192 SHA-1 is the function's complete state at the end of processing an input stream. If the input
193 stream happened to fit neatly into the block size of the hash function, an attacker could extend
194 the input with additional blocks and generate new/unique hash values knowing only the hash
195 output for the original stream. If the secret is at the end of the stream, then attackers are
196 prevented from arbitrarily extending it -- since they have to end the input stream with the
197 password which they don't know. Similarly, if the nonce/created was put at the end, then an
198 attacker could update the nonce to be nonce+created, and add a new created time on the end to
199 generate a new hash.

200

201 The countermeasures above do not cover the case where the token is replayed to a different
202 receiver. There are several (non-normative) possible approaches to counter this threat, which
203 may be used separately or in combination. Their use requires pre-arrangement (possibly in the
204 form of a separately published profile which introduces new password type) among the
205 communicating parties to provide interoperability:

206

- 207 • including the username in the hash, to thwart cases where multiple user accounts
208 have matching passwords (e.g. passwords based on company name)
- 209 • including the domain name in the hash, to thwart cases where the same
210 username/password is used in multiple systems
- 211 • including some indication of the intended receiver in the hash, to thwart cases where
212 receiving systems don't share nonce caches (e.g., two separate application clusters
213 in the same security domain).

214

215 The following illustrates the XML syntax of this element:

216

```
217 <wsse:UsernameToken wsu:Id="Example-1">  
218   <wsse:Username> ... </wsse:Username>  
219   <wsse:Password Type="..."> ... </wsse:Password>  
220   <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>  
221   <wsu:Created> ... </wsu:Created>  
222 </wsse:UsernameToken>
```

223

224 The following describes the attributes and elements listed in the example above:

225

226 /wsse:UsernameToken/wsse:Password

227 This optional element provides password information (or equivalent such as a hash). It is
228 RECOMMENDED that this element only be passed when a secure transport (e.g.
229 HTTP/S) is being used or if the token itself is being encrypted.

230

231 /wsse:UsernameToken/wsse:Password/@Type

232 This optional URI attribute specifies the type of password being provided. The table
233 below identifies the pre-defined types (note that the URI fragments are relative to the URI
234 for this specification).

235

URI	Description
#PasswordText (default)	The actual password for the username, the password hash, or derived password or S/KEY. This type should be used when hashed password equivalents that do not rely on a nonce or creation time are used, or when a digest algorithm other than SHA1 is used.
#PasswordDigest	The digest of the password (and optionally nonce and/or creation timestamp) for the username using the algorithm described above.

236

237 /wsse:UsernameToken/wsse:Password/@{any}

238 This is an extensibility mechanism to allow additional attributes, based on schemas, to be
239 added to the element.

240

241 /wsse:UsernameToken/wsse:Nonce

242 This optional element specifies a cryptographically random nonce. Each message
243 including a <wsse:Nonce> element MUST use a new nonce value in order for web
244 service producers to detect replay attacks.

245

246 /wsse:UsernameToken/wsse:Nonce/@EncodingType

247 This optional attribute URI specifies the encoding type of the nonce (see the definition of
248 <wsse:BinarySecurityToken> for valid values). If this attribute isn't specified then
249 the default of Base64 encoding is used.

250

251 /wsse:UsernameToken/wsu:Created

252 The optional <wsu:Created> element specifies a timestamp used to indicate the
253 creation time. It is defined as part of the <wsu:Timestamp> definition.

254

255 All compliant implementations MUST be able to process the `<wsse:UsernameToken>` element.
256 Where the specification requires that an element be "processed" it means that the element type
257 MUST be recognized to the extent that an appropriate error is returned if the element is not
258 supported.

259

260 Note that `<wsse:KeyIdentifier>` and `<ds:KeyName>` elements as described in the WSS:
261 SOAP Message Security specification are not supported in this profile.

262

263 The following example illustrates the use of this element. In this example the password is sent as
264 clear text and therefore this message should be sent over a confidential channel:

265

```
266 <S11:Envelope xmlns:S11="..." xmlns:wsse="...">
267   <S11:Header>
268     ...
269     <wsse:Security>
270       <wsse:UsernameToken>
271         <wsse:Username>Zoe</wsse:Username>
272         <wsse:Password>IloveDogs</wsse:Password>
273       </wsse:UsernameToken>
274     </wsse:Security>
275     ...
276   </S11:Header>
277   ...
278 </S11:Envelope>
```

279

280 The following example illustrates using a digest of the password along with a nonce and a
281 creation timestamp:

282

```
283 <S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
284   <S11:Header>
285     ...
286     <wsse:Security>
287       <wsse:UsernameToken>
288         <wsse:Username>NNK</wsse:Username>
289         <wsse:Password Type="...#PasswordDigest">
290           weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==
291         </wsse:Password>
292         <wsse:Nonce>WScqanjCEAC4mQoBE07sAQ==</wsse:Nonce>
293         <wsu:Created>2003-07-16T01:24:32Z</wsu:Created>
294       </wsse:UsernameToken>
295     </wsse:Security>
296     ...
297   </S11:Header>
298   ...
299 </S11:Envelope>
```

300

301 **3.2 Token Reference**

302 When a UsernameToken is referenced using `<wsse:SecurityTokenReference>` the
303 `ValueType` attribute is not required. If specified, the value of `#UsernameToken` **MUST** be
304 specified.

305

306 The following encoding formats are pre-defined (note that the URI fragments are relative to the
307 URI for this specification):

308

URI	Description
<code>#UsernameToken</code>	UsernameToken

309

310 When a UsernameToken is referenced from a `<ds:KeyInfo>` element, it can be used to derive
311 a key for a message authentication algorithm using the password. This profile considers specific
312 mechanisms for key derivation to be out of scope. Implementations should agree on a key
313 derivation algorithm in order to be interoperable.

314

315 There is no definition of a KeyIdentifier for a UsernameToken. Consequently, KeyIdentifier
316 references **MUST NOT** be used when referring to a UsernameToken.

317

318 Similarly, there is no definition of a KeyName for a UsernameToken. Consequently, KeyName
319 references **MUST NOT** be used when referring to a UsernameToken.

320

321 All references refer to the `wsu:id` for the token.

322 **3.3 Error Codes**

323 Implementations may use custom error codes defined in private namespaces if needed. But it is
324 **RECOMMENDED** that they use the error handling codes defined in the WSS: SOAP Message
325 Security specification for signature, decryption, and encoding and token header errors to improve
326 interoperability.

327

328 When using custom error codes, implementations should be careful not to introduce security
329 vulnerabilities that may assist an attacker in the error codes returned.

330 **4 Key Derivation**

331 The password associated with a username may be used to derive a shared secret key for the
332 purposes of integrity or confidentiality protecting message contents. This section defines schema
333 extensions and a procedure for deriving such keys. This procedure **MUST** be employed when
334 keys are to be derived from passwords in order to insure interoperability.

335

336 It must be noted that passwords are subject to several kinds of attack, which in turn will lead to
337 the exposure of any derived keys. This key derivation procedure is intended to minimize the risk
338 of attacks on the keys, to the extent possible, but it is ultimately limited by the insecurity of a
339 password that it is possible for a human being to remember and type on a standard keyboard.
340 This is discussed in more detail in the security considerations section of this document.

341

342 Two additional elements are required to enable to derivation of a key from a password. They are
343 <wsse11:Salt> and <wsse11:Iteration>. These values are not secret and MUST be
344 conveyed in the Username token when key derivation is used. When key derivation is used the
345 password MUST NOT be included in the Username token. The receiver will use its knowledge of
346 the password to derive the same key as the sender.

347

348 The following illustrates the syntax of the <wsse11:Salt> and <wsse11:Iteration>
349 elements.

```
350 <wsse:UsernameToken wsse:Id="..." >  
351   <wsse:Username>...</wsse:Username>  
352   <wsse11:Salt>...</wsse11:Salt>  
353   <wsse11:Iteration>...</wsse11:Iteration>  
354 </wsse:UsernameToken>
```

355 The following describes these elements.

356

357 /wsse11:UsernameToken/wsse:Salt

358 This element is combined with the password as described below. Its value is a 128 bit
359 number expressed in hexadecimal. It MUST be present when key derivation is used.

360

361 /wsse11:UsernameToken/wsse11:Iteration

362 This element indicates the number of times the hashing operation is repeated when
363 deriving the key. It is expressed as a decimal value. If it is not present, a value is 1000 is
364 used for the iteration count.

365

366 A key derived from a password may be used either in the calculation of a Message Authentication
367 Code (MAC) or as a symmetric key for encryption. When used in a MAC, the key length will
368 always be 160 bits. When used for encryption, an encryption algorithm MUST NOT be used
369 which requires a key of length greater than 160 bits. A sufficient number of the high order bits of
370 the key will be used for encryption. Unneeded low order bits will be discarded. For example, if the
371 AES-128 algorithm is used, the high order 128 bits will be used and the low order 32 bits will be
372 discarded from the derived 160 bit value.

373

374 The <wsse11:Salt> element is constructed as follows. The high order 8 bits of the Salt will
375 have the value of 01 if the key is to be used in a MAC and 02 if the key is to be used for
376 encryption. The remaining 120 low order bits of the Salt should be a random value.

377

378 The key is derived as follows. The password and Salt are concatenated in that order. Only the
379 actual octets of the password are used, it is not padded or zero terminated. This value is hashed
380 using the SHA1 algorithm. The result of this operation is also hashed using SHA1. This process is
381 repeated until the total number of hash operations equals the Iteration count.

382
383 In other words: $K1 = \text{SHA1}(\text{password} + \text{Salt})$
384 $K2 = \text{SHA1}(K1)$
385 ...
386 $Kn = \text{SHA1}(Kn-1)$

387 Where + means concatenation and n is the iteration count.

388

389 The resulting 160 bit value is used in a MAC function or truncated to the appropriate length for
390 encryption.

391 5 Security Considerations

392 The use of the UsernameToken introduces no additional threats beyond those already identified
393 for other types of SecurityTokens. Replay attacks can be addressed by using message
394 timestamps, nonces, and caching, as well as other application-specific tracking mechanisms.
395 Token ownership is verified by use of keys and man-in-the-middle attacks are generally
396 mitigated. Transport-level security may be used to provide confidentiality and integrity of both the
397 UsernameToken and the entire message body.

398

399 When a password (or password equivalent) in a <UsernameToken> is used for authentication,
400 the password needs to be properly protected. If the underlying transport does not provide enough
401 protection against eavesdropping, the password SHOULD be digested as described in this
402 document. Even so, the password must be strong enough so that simple password guessing
403 attacks will not reveal the secret from a captured message.

404

405 When a password is encrypted, in addition to the normal threats against any encryption, two
406 password-specific threats must be considered: replay and guessing. If an attacker can
407 impersonate a user by replaying an encrypted or hashed password, then learning the actual
408 password is not necessary. One method of preventing replay is to use a nonce as mentioned
409 previously. Generally it is also necessary to use a timestamp to put a ceiling on the number of
410 previous nonces that must be stored. However, in order to be effective the nonce and timestamp
411 must be signed. If the signature is also over the password itself, prior to encryption, then it would
412 be a simple matter to use the signature to perform an offline guessing attack against the
413 password. This threat can be countered in any of several ways including: don't include the
414 password under the signature (the password will be verified later) or sign the encrypted
415 password.

416

417 The reader should also review Section 13 of WSS: SOAP Message Security document for
418 additional discussion on threats and possible counter-measures.

419

420 The security of keys derived from passwords is limited by the attacks available against passwords
421 themselves, such as guessing and brute force. Because of the limited size of password that
422 human beings can remember and limited number of octet values represented by keys that can
423 easily be typed, a typical password represents the equivalent of an entropy source of a maximum
424 of only about 50 bits. For this reason a maximum key size of only 160 bits is supported. Longer
425 keys would simply increase processing without adding to security.

426

427 The key derivation algorithm specified here is based on one described in RFC 2898. It is referred
428 to in that document as PBKDF1. It is used instead of PBKDF2, because it is simpler and keys
429 longer than 160 bits are not required as discussed previously.

430

431 The purpose of the salt is to prevent the bulk pre-computation of key values to be tested against
432 distinct passwords. The Salt value is defined so that MAC and encryption keys are guaranteed to
433 have distinct values even when derived from the same password. This prevents certain
434 cryptanalytic attacks.

435

436 The iteration count is intended to increase the work factor of a guessing or brute force attack, at a
437 minor cost to normal key derivation. An iteration count of at least 1000 (the default) SHOULD
438 always be used.

439

440 This section is non-normative.

441 6 References

441

442 The following are normative references:

- 443 **[SECGLO]** Informational RFC 2828, "Internet Security Glossary," May 2000.
444 **[RFC2119]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels,"
445 RFC 2119, Harvard University, March 1997
446 **[WSS]** OASIS standard, "WSS: SOAP Message Security," TBD.
447 **[SOAP11]** W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.
448 **[SOAP12]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging
449 Framework", 23 June 2003
450 **[URI]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers
451 (URI): Generic Syntax," RFC 3986, MIT/LCS, Day Software, Adobe
452 Systems, January 2005..
453 **[XML-Schema]** W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001.
454 W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.
455 **[XPath]** W3C Recommendation, "XML Path Language", 16 November 1999
456

457 The following are non-normative references included for background and related material:

- 458 **[WS-Security]** OASIS, "Web Services Security: SOAP Message Security" 19 January
459 2004, [http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-](http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0)
460 [soap-message-security-1.0](http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0)
461 **[XML-C14N]** W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001
462 **[EXC-C14N]** W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8
463 July 2002.
464 **[XML-Encrypt]** W3C Working Draft, "XML Encryption Syntax and Processing," 04 March
465 2002
466 W3C Recommendation, "Decryption Transform for XML Signature", 10
467 December 2002.
468 **[XML-ns]** W3C Recommendation, "Namespaces in XML," 14 January 1999.

469 **[XML Signature]** D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-*
470 *Signature Syntax and Processing*, W3C Recommendation, 12 February
471 2002. <http://www.w3.org/TR/xmlsig-core/>
472

Appendix A. Acknowledgements

Contributors:

Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Lab
Merlin	Hughes	Baltimore Technologies
Irving	Reid	Baltimore Technologies
Peter	Dapkus	BEA
Hal	Lockhart	BEA
Steve	Anderson	BMC (Sec)
Srinivas	Davanum	Computer Associates
Thomas	DeMartini	ContentGuard
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Shawn	Sharp	Cyclone Commerce
Ganesh	Vaideeswaran	Documentum
Sam	Wei	Documentum
John	Hughes	Entegrity
Tim	Moses	Entrust
Toshihiro	Nishimura	Fujitsu
Tom	Rutt	Fujitsu
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Paula	Austel	IBM
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Maryann	Hondo	IBM
Michael	McIntosh	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Anthony	Nadalin	IBM
Nataraj	Nagaratnam	IBM
Wayne	Vicknair	IBM
Kelvin	Lawrence	IBM (co-Chair)
Don	Flinn	Individual
Bob	Morgan	Individual
Bob	Atkinson	Microsoft
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Paul	Cotton	Microsoft
Giovanni	Della-Libera	Microsoft
Vijay	Gajjala	Microsoft
Johannes	Klein	Microsoft
Scott	Konersmann	Microsoft
Chris	Kurt	Microsoft
Brian	LaMacchia	Microsoft
Paul	Leach	Microsoft

John	Manferdelli	Microsoft
John	Shewchuk	Microsoft
Dan	Simon	Microsoft
Hervey	Wilson	Microsoft
Chris	Kaler	Microsoft (co-Chair)
Prateek	Mishra	Netegrity
Frederick	Hirsch	Nokia
Senthil	Sengodan	Nokia
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle
Eric	Gravengaard	Reactivity
Stuart	King	Reed Elsevier
Andrew	Nash	RSA Security
Rob	Philpott	RSA Security
Peter	Rostin	RSA Security
Martijn	de Boer	SAP
Blake	Dournaee	Sarvega
Pete	Wenzel	SeeBeyond
Jonathan	Tourzan	Sony
Yassir	Elley	Sun Microsystems
Jeff	Hodges	Sun Microsystems
Ronald	Monzillo	Sun Microsystems
Jan	Alexander	Systinet
Michael	Nguyen	The IDA of Singapore
Don	Adams	TIBCO
Symon	Chang	TIBCO
John	Weiland	US Navy
Phillip	Hallam-Baker	VeriSign
Mark	Hays	Verisign
Hemma	Prafullchandra	VeriSign

475

Appendix B. Revision History

Rev	Date	By Whom	What
WGD 1.1	2004-09-13	Anthony Nadalin	Initial version cloned from the Version 1.0 and Errata
WGD 1.1	2005-05-11	Anthony Nadalin	Issue 373, 388
WGD 1.1	2005-05-17	Anthony Nadalin	Formatting Issues
WGD 1.1	2005-06-14	Anthony Nadalin	Fix Example

