



Web Services Security SOAP Messages with Attachments (SwA) Profile 1.0

Interop 1 Scenarios

Working Draft 04, 21 Oct 2004

Document identifier:

swa-interop1-draft-04.doc

Location:

<http://www.oasis-open.org/committees/wss/>

Editor:

Blake Dournaee, Sarvega Inc. <blake@sarvega.com>

Contributors:

Bruce Rich, IBM <brich@us.ibm.com>

Maneesh Sahu, Actional <maneesh@actional.com>

Frederick Hirsch, Nokia <Frederick.Hirsch@nokia.com>

Abstract:

This document formalizes the interoperability scenarios to be used in the first Web Services Security SwA Profile interoperability event.

Status:

Committee members should send comments on this specification to the wss@lists.oasis-open.org list. Others should subscribe to and send comments to the wss-comment@lists.oasis-open.org list. To subscribe, send an email message to wss-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

Table of Contents

27	Introduction	5
28	1.1 Terminology.....	5
29	2 Test Application	6
30	2.1 Example Ping Element.....	6
31	2.2 Example PingResponse Element.....	6
32	2.3 SOAP Message Packages.....	6
33	3 Scenario #1: Attachment Signature	7
34	3.1 Attachment Properties.....	7
35	3.2 Agreements	7
36	3.2.1 CERT-VALUE	7
37	3.2.2 Signature Trust Root.....	7
38	3.3 Parameters.....	7
39	3.4 General Message Flow	7
40	3.5 First Message – Request	8
41	3.5.1 Message Elements and Attributes	8
42	3.5.2 Message Creation.....	8
43	3.5.3 Responder Message Processing.....	9
44	3.5.4 Example (Non-normative).....	10
45	3.6 Second Message - Response	11
46	3.6.1 Message Elements and Attributes	11
47	3.6.2 Message Creation.....	11
48	3.6.3 Message Processing.....	11
49	3.6.4 Example (Non-normative).....	11
50	3.7 Other processing	11
51	3.7.1 Requester	11
52	3.7.2 Responder	12
53	3.8 Expected Security Properties.....	12
54	4 Scenario #2 – Attachment Encryption	13
55	4.1 Attachment Properties.....	13
56	4.2 Agreements	13
57	4.2.1 CERT-VALUE	13
58	4.2.2 Signature Trust Root.....	13
59	4.3 Parameters.....	13
60	4.4 General Message Flow	13
61	4.5 First Message - Request.....	14
62	4.5.1 Message Elements and Attributes	14
63	4.5.2 Message Creation.....	14
64	4.5.3 Responder Message Processing.....	16
65	4.5.4 Example (Non-normative).....	16
66	4.6 Second Message - Response	17

67	4.6.1 Message Elements and Attributes	17
68	4.6.2 Message Creation.....	18
69	4.6.3 Message Processing.....	18
70	4.6.4 Example (Non-normative)	18
71	4.7 Other processing	18
72	4.7.1 Requester	18
73	4.7.2 Responder	18
74	4.8 Expected Security Properties.....	18
75	5 Scenario #3 – Attachment Signature and Encryption.....	19
76	5.1 Attachment Properties.....	19
77	5.2 Agreements.....	19
78	5.2.1 CERT-VALUE	19
79	5.2.2 Signature Trust Root.....	19
80	5.3 Parameters.....	19
81	5.4 General Message Flow	20
82	5.5 First Message - Request.....	20
83	5.5.1 Message Elements and Attributes	20
84	5.5.2 Message Creation.....	21
85	5.5.3 Responder Message Processing.....	23
86	5.5.4 Example (Non-normative)	23
87	5.6 Second Message - Response	25
88	5.6.1 Message Elements and Attributes	25
89	5.6.2 Message Creation.....	25
90	5.6.3 Message Processing.....	26
91	5.6.4 Example (Non-normative)	26
92	5.7 Other processing	26
93	5.7.1 Requester	26
94	5.7.2 Responder	26
95	5.8 Expected Security Properties.....	26
96	6 Scenario #4 – Attachment Signature and Encryption with MIME Headers	27
97	6.1 Attachment Properties.....	27
98	6.2 Agreements.....	27
99	6.2.1 CERT-VALUE	27
100	6.2.2 Signature Trust Root.....	27
101	6.3 Parameters.....	27
102	6.4 General Message Flow	28
103	6.5 First Message - Request.....	28
104	6.5.1 Message Elements and Attributes	28
105	6.5.2 Message Creation.....	29
106	6.5.3 Responder Message Processing.....	31
107	6.5.4 Example (Non-normative)	31
108	6.6 Second Message - Response	33
109	6.6.1 Message Elements and Attributes	33

110	6.6.2 Message Creation.....	33
111	6.6.3 Message Processing.....	34
112	6.6.4 Example (Non-normative).....	34
113	6.7 Other processing.....	34
114	6.7.1 Requester	34
115	6.7.2 Responder	34
116	6.8 Expected Security Properties.....	34
117	7 References.....	35
118	7.1 Normative	35
119	Appendix A. Ping Application WSDL File	36
120	Appendix B. Revision History	38
121	Appendix C. Notices	39
122		

123 Introduction

124 This document describes the message exchanges to be tested during the first interoperability
125 event of the Web Services Security SOAP Message with Attachments Profile. All scenarios use
126 the Request/Response Message Exchange Pattern (MEP) with no intermediaries. All scenarios
127 invoke the same simple application. To avoid confusion, they are called Scenario #1 through
128 Scenario #4.

129 These scenarios are intended to test the interoperability of different implementations performing
130 common operations and to test the soundness of the various specifications and clarity and mutual
131 understanding of their meaning and proper application.

132 THESE SCENARIOS ARE NOT INTENDED TO REPRESENT REASONABLE OR USEFUL
133 PRACTICAL APPLICATIONS OF THE SPECIFICATIONS. THEY HAVE BEEN DESIGNED
134 PURELY FOR THE PURPOSES INDICATED ABOVE AND DO NOT NECESSARILY
135 REPRESENT EFFICIENT OR SECURE MEANS OF PERFORMING THE INDICATED
136 FUNCTIONS. IN PARTICULAR THESE SCENARIOS ARE KNOWN TO VIOLATE SECURITY
137 BEST PRACTICES IN SOME RESPECTS AND IN GENERAL HAVE NOT BEEN EXTENSIVELY
138 VETTED FOR ATTACKS.

139 1.1 Terminology

140 The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*,
141 and *optional* in this document are to be interpreted as described in [RFC2119].

142 2 Test Application

143 All four scenarios use the same, simple application.

144 The Requester sends a Ping element with a value of a string as the single child to a SOAP
145 request. The value should be the name of the organization that has developed the software and
146 the number of the scenario, e.g. "Acme Corp. – Scenario #1".

147 The Responder returns a PingResponse element with a value of the same string.

148 Each interaction will also include a SOAP attachment secured via one of the content level
149 security mechanisms described in **[WSS-SwA]**. For the purpose of these interoperability
150 scenarios, the Ping request and response elements will not have security properties applied to
151 them; they are used only to keep track of the specific scenarios.

152 2.1 Example Ping Element

```
153 <Ping xmlns="http://xmlsoap.org/Ping">  
154   <text>Acme Corp. - Scenario #1</text>  
155 </Ping>
```

156 2.2 Example PingResponse Element

```
157 <PingResponse xmlns="http://xmlsoap.org/Ping">  
158   <text> Acme Corp. - Scenario #1</text>  
159 </PingResponse>
```

160 2.3 SOAP Message Packages

161 When SOAP attachments are used as specified in **[SwA]** the main SOAP payload is
162 accompanied by a MIME header and possibly multiple boundary parts. This is known as a SOAP
163 message package. All interoperability scenarios in this document assume that a proper SOAP
164 message package is constructed using the MIME headers appropriate to **[SwA]**. Interoperability
165 of the SOAP message package format, including the appropriate use of the MIME header and
166 boundary semantics, is outside the scope of this interoperability document.

167 2.4 URI Shorthand Notation

168 For brevity, the following shorthand is used in describing URI strings:

URI	Shorthand
http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform	#Attachment-Content-Only-Transform
http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete-Transform	#Attachment-Complete-Transform

169

170 **3 Scenario #1: Attachment Signature**

171 Scenario #1 tests the interoperability of a signed attachment using an X.509 certificate. The
172 certificate used to verify the signature shall be present in the SOAP header. No security
173 properties are applied to any part of the SOAP envelope..

174 **3.1 Attachment Properties**

175 This section specifies the attachment properties BEFORE security operations are applied. The
176 Content-Type of the attachment MUST be image/jpeg. The Content-Transfer-Encoding MUST be
177 base64. The attachment MUST have a Content-Id header that uniquely identifies the attachment.
178 The generation of the Content-Id header is out of scope.

179 **3.2 Agreements**

180 This section describes the agreements that must be made, directly or indirectly between parties
181 who wish to interoperate.

182 **3.2.1 CERT-VALUE**

183 This is an opaque identifier indicating the X.509 certificate to be used. The certificate in question
184 MUST be obtained by the Requester by unspecified means. The certificate SHOULD NOT have a
185 KeyUsage extension. If it does contain a KeyUsage extension, it SHOULD include the value of
186 digitalSignature.

187 **3.2.2 Signature Trust Root**

188 This refers generally to agreeing on at least one trusted key and any other certificates and
189 sources of revocation information sufficient to validate certificates sent for the purpose of
190 signature verification.

191 **3.3 Parameters**

192 This section describes parameters that are required to correctly create or process messages, but
193 not a matter of mutual agreement.

194 No parameters are required.

195 **3.4 General Message Flow**

196 This section provides a general overview of the flow of messages.

197 This contract covers a request/response MEP over the HTTP binding. SOAP 1.1 MUST be used.
198 The SOAP envelope MUST be wrapped in a SOAP Message Package as specified by **[SwA]**. As
199 required by SOAP 1.1, the SOAPAction HTTP header MUST be present. Any value, including a
200 null string may be used. The recipient SHOULD ignore the value. The request contains a signed
201 attachment. The certificate used for signing is included in the message.

202 The Responder verifies the signature over the attachment. If no errors are detected it returns the
203 response with no additional security properties.

204 3.5 First Message – Request

205 3.5.1 Message Elements and Attributes

206 Elements not listed in the following table MAY be present, but MUST NOT be marked with the
207 mustUnderstand="1" attribute. Items marked mandatory MUST be generated and processed.
208 Items marked optional MAY be generated and MUST be processed if present. Items MUST
209 appear in the order specified, except as noted.

210

Name	Mandatory?
Envelope	Mandatory
Header	Mandatory
Security	Mandatory
mustUnderstand="1"	Mandatory
BinarySecurityToken	Mandatory
Signature	Mandatory
SignedInfo	Mandatory
CanonicalizationMethod	Mandatory
SignatureMethod	Mandatory
Reference	Mandatory
Transforms	Mandatory
Transform	Mandatory
SignatureValue	Mandatory
KeyInfo	Mandatory
Body	Mandatory
Ping	Mandatory

211 3.5.2 Message Creation

212 3.5.2.1 Security

213 The Security element MUST contain the mustUnderstand="1" attribute.

214 3.5.2.2 BinarySecurityToken

215 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
216 be labeled with an Id so it can be referenced by the signature. The value MUST be a public key
217 certificate suitable for verifying the signature. The certificate SHOULD NOT have a KeyUsage
218 extension. If it does contain a KeyUsage extension, it SHOULD include the value of
219 digitalSignature. The Requester must have access to the private key corresponding to the public
220 key in the certificate.

221 **3.5.2.3 Signature**

222 The signature is over the attachment content only, using the #Attachment-Content-Only-
223 Transform

224 **3.5.2.3.1 SignedInfo**

225 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST
226 be RSA-SHA1.

227 **3.5.2.3.2 Reference**

228 The Reference MUST specify a URI using the cid scheme that points to the Content-Id of the
229 attachment. The only Transform specified MUST be #Attachment-Content-Only. The
230 DigestMethod MUST be SHA1.

231 **3.5.2.3.3 SignatureValue**

232 The SignatureValue MUST be calculated as specified by the specification, using the private key
233 corresponding to the public key specified in the certificate in the BinarySecurityToken.

234 **3.5.2.3.4 KeyInfo**

235 The KeyInfo MUST contain a SecurityTokenReference with a reference to a relative URI which
236 indicates the BinarySecurityToken containing the certificate which will be used for signature
237 verification.

238 **3.5.2.4 Post Operation Attachment Properties**

239 This section specifies the attachment properties AFTER security operations are applied. The
240 Content-Type of the attachment MUST be image/jpeg. The Content-Transfer-Encoding MUST be
241 base64. The attachment MUST have a Content-Id header that uniquely identifies the attachment.
242 The generation of the Content-Id header is out of scope.

243 **3.5.3 Responder Message Processing**

244 This section describes the processing performed by the Responder. If an error is detected, the
245 Responder MUST cease processing the message and issue a Fault with a value of
246 FailedAuthentication.

247 **3.5.3.1 Security**

248 **3.5.3.2 BinarySecurityToken**

249 The certificate in the token MUST be validated. The Subject of the certificate MUST be an
250 authorized entity. The public key in the certificate MUST be retained for verification of the
251 signature.

252 **3.5.3.3 Signature**

253 The attachment MUST be verified against the signature using the specified algorithms and
254 transforms and the retained public key.

255 **3.5.3.4 Attachment**

256 After the attachment's signature has been verified, it should be passed to the application.

257

3.5.4 Example (Non-normative)

258

```
Content-Type: multipart/related; boundary="sig-example"; type="text/xml"
```

259

```
--sig-example
```

260

```
Content-Type: text/xml; charset=utf-8
```

261

```
<?xml version="1.0" encoding="utf-8" ?>
```

262

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
```

263

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

264

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

265

```
  <soap:Header>
```

266

```
    <wsse:Security soap:mustUnderstand="1"
```

267

```
      xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
```

268

```
        <!-- This is the certificate used to verify the signature -->
```

269

```
        <wsse:BinarySecurityToken ValueType="wsse:X509v3"
```

270

```
          EncodingType="wsse:Base64Binary" xmlns:wsu="http://docs.oasis
```

271

```
          open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
```

272

```
          wsu:Id="mySigCert">MII...hk</wsse:BinarySecurityToken>
```

273

```
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

274

```
          <SignedInfo>
```

275

```
            <CanonicalizationMethod
```

276

```
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

277

```
            <SignatureMethod
```

278

```
              Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
```

279

```
            <Reference URI="cid:signature">
```

280

```
              <Transforms>
```

281

```
                <Transform Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-
```

282

```
                2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform"/>
```

283

```
                </Transforms>
```

284

```
                <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

285

```
                <DigestValue>QTV...dw</DigestValue>
```

286

```
              </Reference>
```

287

```
            </SignedInfo>
```

288

```
          <SignatureValue>H+x0...gUw</SignatureValue>
```

289

```
          <KeyInfo>
```

290

```
            <wsse:SecurityTokenReference>
```

291

```
              <wsse:Reference URI="#mySigCert" />
```

292

```
            </wsse:SecurityTokenReference>
```

293

```
          </KeyInfo>
```

294

```
        </Signature>
```

295

```
      </wsse:Security>
```

296

```
    </soap:Header>
```

297

```
    <soap:Body>
```

298

```
      <Ping xmlns="http://xmlsoap.org/Ping">
```

299

```
        <text>Acme Corp. - Scenario #1</text>
```

300

```
      </Ping>
```

301

```
    </soap:Body>
```

302

```
  </soap:Envelope>
```

303

```
--sig-example
```

304

```
Content-Type: image/jpeg
```

305

```
Content-Id: <signature>
```

306

```
Content-Transfer-Encoding: base64
```

307

```
Dcg3AdGFcFs3764fddSArk
```

308

309

310

311

312

313

314

315 **3.6 Second Message - Response**

316 **3.6.1 Message Elements and Attributes**

317 Items not listed in the following table MUST NOT be created or processed. Items marked
318 mandatory MUST be generated and processed. Items marked optional MAY be generated and
319 MUST be processed if present. Items MUST appear in the order specified, except as noted.

320

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

321

322 **3.6.2 Message Creation**

323 **3.6.2.1 Security**

324 There are no security properties on the response message

325 **3.6.2.2 Body**

326 The body element MUST be not be signed or encrypted

327 **3.6.3 Message Processing**

328 The response is passed to the application without modification.

329 **3.6.4 Example (Non-normative)**

330 Here is an example response.

```
331 <?xml version="1.0" encoding="utf-8" ?>  
332 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
333 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
334 xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
335 <soap:Body>  
336 <PingResponse xmlns="http://xmlsoap.org/Ping">  
337 <text> Acme Corp. - Scenario #1</text>  
338 </PingResponse>  
339 </soap:Body>  
340 </soap:Envelope>
```

341

342 **3.7 Other processing**

343 This section describes processing that occurs outside of generating or processing a message.

344 **3.7.1 Requester**

345 No additional processing is required.

346 **3.7.2 Responder**

347 No additional processing is required.

348 **3.8 Expected Security Properties**

349 Use of the service is restricted to authorized parties that sign the attachment. The attachment of
350 the request is protected against modification and interception. The response does not have any
351 security properties.

352 4 Scenario #2 – Attachment Encryption

353 The SOAP request has an attachment that has been encrypted. The encryption is done using a
354 symmetric cipher. The symmetric encryption key is further encrypted for a specific recipient
355 identified by an X.509 certificate. The certificate associated with the key encryption is provided to
356 the requestor out-of-band. No security properties are applied to any part of the SOAP envelope.

357 4.1 Attachment Properties

358 This section specifies the attachment properties BEFORE security operations are applied. The
359 Content-Type of the attachment MUST be image/jpeg. The Content-Transfer-Encoding MUST be
360 base64. The attachment MUST have a Content-Id header that uniquely identifies the attachment.
361 The generation of the Content-Id header is out of scope.

362 4.2 Agreements

363 This section describes the agreements that must be made, directly or indirectly between parties
364 who wish to interoperate.

365 4.2.1 CERT-VALUE

366 This is an opaque identifier indicating the X.509 certificate to be used. The certificate in question
367 MUST be obtained by the Requester by unspecified means. The certificate SHOULD NOT have a
368 KeyUsage extension. If it does contain a KeyUsage extension, it SHOULD include the values of
369 keyEncipherment.

370 The Responder MUST have access to the Private key corresponding to the Public key in the
371 certificate.

372 4.2.2 Signature Trust Root

373 There is no digital signature operation for this scenario

374 4.3 Parameters

375 This section describes parameters that are required to correctly create or process messages, but
376 not a matter of mutual agreement.

377 No parameters are required.

378 4.4 General Message Flow

379 This section provides a general overview of the flow of messages.

380 This contract covers a request/response MEP over the HTTP binding. SOAP 1.1 MUST be used.
381 The SOAP envelope MUST be wrapped in a SOAP Message Package as specified by **[SwA]**.
382 The Content-Transfer-Encoding for the encrypted attachment MUST be base64. As required by
383 SOAP 1.1, the SOAPAction HTTP header MUST be present. Any value, including a null string
384 may be used. The recipient SHOULD ignore the value. The request contains an encrypted SOAP
385 attachment. The attachment is encrypted with a random symmetric key, which is encrypted using
386 a public key certificate. The certificate used for the encryption is provided to the Requestor out of
387 band. The Responder decrypts the attachment using the symmetric key which is decrypted with
388 the matching private key. If no errors are detected it returns the response without any security
389 properties. If there is a decryption failure a fault is returned as outlined in section 4.5.3.

390 **4.5 First Message - Request**

391 **4.5.1 Message Elements and Attributes**

392 Items not listed in the following table MAY be present, but MUST NOT be marked with the
393 mustUnderstand="1" attribute. Items marked mandatory MUST be generated and processed.
394 Items marked optional MAY be generated and MUST be processed if present. Items MUST
395 appear in the order specified, except as noted.

396

Name	Mandatory?
Envelope	Mandatory
Header	Mandatory
Security	Mandatory
mustUnderstand="1"	Mandatory
BinarySecurityToken	Mandatory
EncryptedKey	Mandatory
EncryptionMethod	Mandatory
KeyInfo	Mandatory
SecurityTokenReference	Mandatory
CipherData	Mandatory
ReferenceList	Mandatory
EncryptedData	Mandatory
EncryptionMethod	Mandatory
CipherData	Mandatory
CipherReference	Mandatory
Transforms	Mandatory
Transform	Mandatory
Body	Mandatory
Ping	Mandatory

397 **4.5.2 Message Creation**

398 **4.5.2.1 Security**

399 The Security element MUST contain the mustUnderstand="1" attribute.

400 **4.5.2.2 BinarySecurityToken**

401 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
402 be labeled with an Id so it can be referenced EncryptedKey security token reference. The value
403 MUST be a Public Key certificate suitable for symmetric key encryption. The certificate SHOULD
404 NOT have a KeyUsage extension. If it does contain a KeyUsage extension, it SHOULD include
405 the values of keyEncipherment and dataEncipherment. The Responder must have access to the
406 private key corresponding to the public key in the certificate.

407 **4.5.2.3 EncryptedKey**

408 The EncryptionMethod MUST contain the Algorithm attribute. The algorithm MUST be RSA v1.5.
409 The KeyInfo MUST contain a SecurityTokenReference with a reference to a relative URI which
410 indicates the BinarySecurityToken containing the certificate which will be used to decrypt the
411 symmetric key.
412 The CipherData MUST contain the encrypted form of the random key, encrypted under the Public
413 Key specified in the specified X.509 certificate, using the specified algorithm.
414 The ReferenceList MUST contain a DataReference which has the value of a relative URI that
415 refers to the EncryptedData element that refers to the encrypted attachment.

416 **4.5.2.4 EncryptedData**

417 The EncryptedData element refers to the encrypted attachment. The Type attribute MUST be
418 present and it MUST have a value of #Attachment-Content-Only. The EncryptedData element
419 MUST be referenced by the ReferenceList element in the EncryptedKey element. The
420 EncryptedData MUST have a MimeType attribute with the value of image/jpeg.

421 **4.5.2.5 EncryptionMethod**

422 The encryption method MUST be Triple-DES in CBC mode.

423 **4.5.2.6 CipherData**

424 The CipherData MUST refer to the encrypted attachment with a CipherReference element. The
425 CipherReference element MUST refer to the attachment using a URI with a cid scheme. The
426 CipherReference must have a single Transforms element with a single Transform child with an
427 Algorithm attribute value of #Attachment-Content-Only-Transform.

428

429 **4.5.2.7 Body**

430 The body element MUST not have any security operations applied to it.

431 **4.5.2.8 Ping**

432 The Ping element should contain the scenario number and the name of the entity performing the
433 request.

434 **4.5.2.9 Post Operation Attachment Properties**

435 This section specifies the attachment properties AFTER security operations are applied. The
436 Content-Type of the attachment MUST be application/octet-stream. The Content-Transfer-
437 Encoding MUST be base64. The attachment MUST have a Content-Id header that uniquely
438 identifies the attachment. The generation of the Content-Id header is out of scope. The Content-Id
439 MUST match the Content-Id before encryption.

440 4.5.3 Responder Message Processing

441 This section describes the processing performed by the Responder. If an error is detected, the
442 Responder MUST cease processing the message and issue a Fault with a value of
443 FailedDecryption.

444 4.5.3.1 Security

445 4.5.3.2 BinarySecurityToken

446 The public key in the certificate MUST be used to decrypt the symmetric key. No trust validation
447 of the public key is required. The responder MUST have the matching private key.

448 4.5.3.3 EncryptedKey

449 The random key contained in the CipherData MUST be decrypted using the private key
450 corresponding to the certificate specified by the SecurityTokenReference, using the specified
451 algorithm.

452 4.5.3.4 EncryptedData

453 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted
454 symmetric key.

455 4.5.3.5 Attachment

456 After decrypting the attachment, it should be passed to the application

457 4.5.4 Example (Non-normative)

458 Here is an example request.

```
459 Content-Type: multipart/related; boundary="enc-example"; type="text/xml"
460 --enc-example
461 Content-Type: text/xml; charset=utf-8
462
463 <?xml version="1.0" encoding="utf-8" ?>
464 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
465 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
466 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
467 <soap:Header>
468 <wsse:Security soap:mustUnderstand="1"
469 xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
470 secext-1.0.xsd">
471
472 <!-- This certificate is used for symmetric key encryption -->
473 <wsse:BinarySecurityToken
474 Value="MIIE...hk"
475 EncodingType="wsse:Base64Binary"
476 xmlns:wsu="http://docs.oasis
477 open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
478 wsu:Id="myEncCert">MIIE...hk</wsse:BinarySecurityToken>
479
480 <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
481 <xenc:EncryptionMethod
482 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
483 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
484 <wsse:SecurityTokenReference>
485 <wsse:Reference URI="#myEncCert" />
486 </wsse:SecurityTokenReference>
487 </KeyInfo>
488 <xenc:CipherData>
489 <xenc:CipherValue>dNYS...fQ</xenc:CipherValue>
```



```

490     </xenc:CipherData>
491     <xenc:ReferenceList>
492       <xenc:DataReference URI="#encrypted-attachment" />
493     </xenc:ReferenceList>
494   </xenc:EncryptedKey>
495
496   <!-- The EncryptedData portion here refers to content of the attachment -->
497
498   <xenc:EncryptedData wsu:Id="encrypted-attachment"
499     Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
500 1.0#Attachment-Content-Only" MimeType="image/jpeg">
501     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
502     <xenc:EncryptionMethod
503       Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
504     <xenc:CipherData>
505       <xenc:CipherReference URI="cid:enc">
506         <xenc:Transforms>
507           <ds:Transform
508             Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
509 profile-1.0#Attachment-Content-Only-Transform" />
510           <ds:Transform
511             />
512         </xenc:Transforms>
513       </xenc:CipherReference>
514     </xenc:CipherData>
515   </xenc:EncryptedData>
516
517 </wsse:Security>
518 </soap:Header>
519 <soap:Body>
520   <Ping xmlns="http://xmlsoap.org/Ping">
521     <text>Acme Corp. - Scenario #2</text>
522   </Ping>
523 </soap:Body>
524 </soap:Envelope>
525 --enc-example
526 Content-Type: application/octet-stream
527 Content-Id: <enc>
528 Content-Transfer-Encoding: base64
529 Dsh5SA3thsRh3Dh54wafDhjaq2

```

530

531 4.6 Second Message - Response

532 4.6.1 Message Elements and Attributes

533 Items not listed in the following table MUST NOT be created or processed. Items marked
534 mandatory MUST be generated and processed. Items marked optional MAY be generated and
535 MUST be processed if present. Items MUST appear in the order specified, except as noted.

536

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

537

538 **4.6.2 Message Creation**

539 The response message MUST NOT contain a <wsse:Security> header. Any other header
540 elements MUST NOT be labeled with a mustUnderstand="1" attribute.

541 **4.6.2.1 Security**

542 There are no security properties on the response message

543 **4.6.2.2 Body**

544 The body element MUST be not be signed or encrypted

545 **4.6.3 Message Processing**

546 The response is passed to the application without modification.

547 **4.6.4 Example (Non-normative)**

548 Here is an example response.

```
549 <?xml version="1.0" encoding="utf-8" ?>  
550 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
551 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
552 xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
553 <soap:Body>  
554 <PingResponse xmlns="http://xmlsoap.org/Ping">  
555 <text> Acme Corp. - Scenario #2</text>  
556 </PingResponse>  
557 </soap:Body>  
558 </soap:Envelope>
```

559 **4.7 Other processing**

560 This section describes processing that occurs outside of generating or processing a message.

561 **4.7.1 Requester**

562 No additional processing is required.

563 **4.7.2 Responder**

564 No additional processing is required.

565 **4.8 Expected Security Properties**

566 The attachment content is private for the holder of the appropriate private key. There should be
567 no inferences made regarding the authenticity of the sender. The response is not protected in any
568 way.

569 **5 Scenario #3 – Attachment Signature and**
570 **Encryption**

571 The SOAP request contains an attachment that has been signed and then encrypted. The
572 certificate associated with the encryption is provided out-of-band to the requestor. The certificate
573 used to verify the signature is provided in the header. The Response Body is not signed or
574 encrypted. There are two certificates in the request message. One identifies the recipient of the
575 encrypted attachment and one identifies the signer.

576 **5.1 Attachment Properties**

577 This section specifies the attachment properties BEFORE security operations are applied. The
578 Content-Type of the attachment MUST be text/xml. The Content-Transfer-Encoding MUST be
579 8bit ASCII (8bit). The attachment MUST have a Content-Id header that uniquely identifies the
580 attachment. The generation of the Content-Id header is out of scope. An example of what the
581 attachment may look like before encryption and signing is shown as follows. This example is non-
582 normative.

```
583 --enc-sig-example  
584 Content-Type: text/xml; charset=utf-8  
585 Content-Transfer-Encoding: 8bit  
586 Content-ID: <encsignexample>  
587  
588 <?xml version=1.0" encoding="utf-8"?>  
589 <somexml/>
```

590

591 **5.2 Agreements**

592 This section describes the agreements that must be made, directly or indirectly between parties
593 who wish to interoperate.

594 **5.2.1 CERT-VALUE**

595 This is an opaque identifier indicating the X.509 certificate to be used. The certificate in question
596 MUST be obtained by the Requester by unspecified means. The certificate SHOULD NOT have a
597 KeyUsage extension. If it does contain a KeyUsage extension, it SHOULD include the values of
598 keyEncipherment, dataEncipherment and digitalSignature.

599 The Responder MUST have access to the private key corresponding to the public key in the
600 certificate.

601 **5.2.2 Signature Trust Root**

602 This refers generally to agreeing on at least one trusted key and any other certificates and
603 sources of revocation information sufficient to validate certificates sent for the purpose of
604 signature verification.

605 **5.3 Parameters**

606 This section describes parameters that are required to correctly create or process messages, but
607 not a matter of mutual agreement.

608 No parameters are required.

609 5.4 General Message Flow

610 This section provides a general overview of the flow of messages.

611 This contract covers a request/response MEP over the HTTP binding. SOAP 1.1 MUST be used.

612 The SOAP envelope MUST be wrapped in a SOAP Message Package as specified by [SwA].

613 The Content-Transfer-Encoding for the encrypted attachment MUST be base64. As required by

614 SOAP 1.1, the SOAPAction HTTP header MUST be present. Any value, including a null string

615 may be used. The recipient SHOULD ignore the value. The request contains an attachment,

616 which is signed and then encrypted. The certificate for encryption is provided externally to the

617 requestor but conveyed in the request message. The attachment is encrypted with a random

618 symmetric key that is encrypted with a public key certificate. The certificate for signing is included

619 in the message. The Responder decrypts the attachment using its private key and then verifies

620 the signature using the included public key certificate. If no errors are detected it returns the

621 Response with no security properties.

622 5.5 First Message - Request

623 5.5.1 Message Elements and Attributes

624 Items not listed in the following table MAY be present, but MUST NOT be marked with the

625 mustUnderstand="1" attribute. Items marked mandatory MUST be generated and processed.

626 Items marked optional MAY be generated and MUST be processed if present. Items MUST

627 appear in the order specified, except as noted.

628

Name	Mandatory?
Envelope	Mandatory
Header	Mandatory
Security	Mandatory
mustUnderstand="1"	Mandatory
BinarySecurityToken	Mandatory
EncryptedKey	Mandatory
EncryptionMethod	Mandatory
KeyInfo	Mandatory
SecurityTokenReference	Mandatory
CipherData	Mandatory
ReferenceList	Mandatory
EncryptedData	Mandatory
EncryptionMethod	Mandatory
CipherData	Mandatory
CipherReference	Mandatory

Transforms	Mandatory
Transform	Mandatory
BinarySecurityToken	Mandatory
Signature	Mandatory
SignedInfo	Mandatory
CanonicalizationMethod	Mandatory
SignatureMethod	Mandatory
Reference	Mandatory
Transforms	Mandatory
Transform	Mandatory
SignatureValue	Mandatory
KeyInfo	Mandatory
Body	Mandatory
Ping	Mandatory

629 **5.5.2 Message Creation**

630 **5.5.2.1 Security**

631 The Security element MUST contain the mustUnderstand="1" attribute.

632 **5.5.2.2 BinarySecurityToken**

633 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
634 be labeled with an Id so it can be uniquely referenced. The value MUST be a PK certificate
635 suitable for encrypting the content. The certificate SHOULD NOT have a KeyUsage extension. If
636 it does contain a KeyUsage extension, it SHOULD include the value of keyEncipherment and
637 dataEncipherment.

638 **5.5.2.3 EncryptedKey**

639 The EncryptionMethod MUST contain the Algorithm attribute. The algorithm MUST be RSA v1.5.

640 The KeyInfo MUST contain a SecurityTokenReference with a Reference child that points to the
641 X.509 certificate of the recipient. The Reference child should point to a relative URI which
642 indicates the BinarySecurityToken containing the certificate which will be used to decrypt the
643 symmetric key.

644 The CipherData MUST contain the encrypted form of the random key, encrypted under the Public
645 Key specified in the specified X.509 certificate, using the specified algorithm.

646 The ReferenceList MUST contain a DataReference which has the value of a relative URI that
647 refers to the EncryptedData element that refers to the encrypted attachment.

648 **5.5.2.4 EncryptedData**

649 The EncryptedData element refers to the encrypted attachment. The Type attribute MUST be
650 present and it MUST have a value of #Attachment-Content-Only. The EncryptedData element
651 MUST be referenced by the ReferenceList element in the EncryptedKey element. The
652 EncryptedData MUST have a MimeType attribute with the value of text/xml.

653 **5.5.2.5 EncryptionMethod**

654 The encryption method MUST be Triple-DES in CBC mode.

655 **5.5.2.6 CipherData**

656 The CipherData MUST refer to the encrypted attachment with a CipherReference element. The
657 CipherReference element MUST refer to the attachment using a URI with a cid scheme. The
658 CipherReference must have a Transforms child with a single Transform sub child with the value
659 of #Attachment-Content-Only-Transform.

660 **5.5.2.7 BinarySecurityToken**

661 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
662 be labeled with an Id so it can be referenced by the signature. The value MUST be a PK
663 certificate suitable for verifying the signature. The certificate SHOULD NOT have a KeyUsage
664 extension. If it does contain a KeyUsage extension, it SHOULD include the values of
665 digitalSignature. The Requester must have access to the private key corresponding to the public
666 key in the certificate.

667 **5.5.2.8 Signature**

668 The signature is over the attachment content only, using the #Attachment-Content-Only-
669 Transform

670 **5.5.2.8.1 SignedInfo**

671 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST
672 be RSA-SHA1.

673 **5.5.2.8.2 Reference**

674 The Reference MUST specify a URI using the cid scheme that points to the Content-Id of the
675 attachment. The only Transform specified MUST be #Attachment-Content-Only. The
676 DigestMethod MUST be SHA1.

677 **5.5.2.8.3 SignatureValue**

678 The SignatureValue MUST be calculated as specified by the specification, using the private key
679 corresponding to the public key specified in the certificate in the BinarySecurityToken.

680 **5.5.2.8.4 KeyInfo**

681 The KeyInfo MUST contain a SecurityTokenReference with a reference to a relative URI which
682 indicates the BinarySecurityToken containing the certificate which will be used for signature
683 verification.

684 **5.5.2.9 Body**

685 The contents of the body MUST NOT be encrypted or signed

686 **5.5.2.10 Post Operation Attachment Properties**

687 This section specifies the attachment properties AFTER security operations are applied. The
688 Content-Type of the attachment MUST be application/octet-stream. The Content-Transfer-
689 Encoding MUST be base64. The attachment MUST have a Content-Id header that uniquely
690 identifies the attachment. The generation of the Content-Id header is out of scope. The Content-Id
691 MUST match the Content-Id before encryption.

692 **5.5.3 Responder Message Processing**

693 This section describes the processing performed by the Responder. If an error is detected, the
694 Responder MUST cease processing the message and issue a Fault with a value of
695 FailedAuthentication.

696 **5.5.3.1 Security**

697 **5.5.3.2 BinarySecurityToken**

698 The public key in the certificate MUST be used to decrypt the symmetric key. No trust validation
699 of the public key is required. The responder MUST have the matching private key.

700 **5.5.3.3 EncryptedKey**

701 The random key contained in the CipherData MUST be decrypted using the private key
702 corresponding to the certificate specified by the SecurityTokenReference, using the specified
703 algorithm.

704 **5.5.3.4 EncryptedData**

705 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted
706 symmetric key.

707 **5.5.3.5 Attachment**

708 After decrypting the attachment, it should have its signature verified

709 **5.5.3.6 BinarySecurityToken**

710 The certificate in the token MUST be validated. The Subject of the certificate MUST be an
711 authorized entity. The public key in the certificate MUST be retained for verification of the
712 signature.

713 **5.5.3.7 Signature**

714 The attachment MUST be verified against the signature using the specified algorithms and
715 transforms and the retained public key.

716 **5.5.3.8 Attachment**

717 After the attachment's signature has been verified, it should be passed to the application

718 **5.5.4 Example (Non-normative)**

719 Here is an example request.

```
720 Content-Type: multipart/related; boundary="enc-sig-example"; type="text/xml"  
721 --enc-sig-example  
722 Content-Type: text/xml; charset=utf-8
```

723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1"
xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">

      <!-- This certificate is used for symmetric key encryption -->
      <wsse:BinarySecurityToken
        ValueType="wsse:X509v3"
        EncodingType="wsse:Base64Binary"
        xmlns:wsu="http://docs.oasis
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="myEncCert">MII...hk</wsse:BinarySecurityToken>

      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#myEncCert" />
          </wsse:SecurityTokenReference>
        </KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>dNYS...fQ=</xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#encrypted-signed-attachment" />
        </xenc:ReferenceList>
      </xenc:EncryptedKey>

      <!-- The EncryptedData portion here refers to content of the attachment -->

      <xenc:EncryptedData wsu:Id="encrypted-signed-attachment"
        Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
1.0#Attachment-Content-Only" MimeType="text/xml">
        xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
        <xenc:CipherData>
          <xenc:CipherReference URI="cid:encsignexample">
            <xenc:Transforms>
              <ds:Transform
                Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
profile-1.0#Attachment-Content-Only-Transform" />
            </xenc:Transforms>
          </xenc:CipherReference>
        </xenc:CipherData>
      </xenc:EncryptedData>

      <!-- This certificate is used to verify the signature -->
      <wsse:BinarySecurityToken ValueType="wsse:X509v3"
        EncodingType="wsse:Base64Binary" xmlns:wsu="http://docs.oasis
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="mySigCert">MII...hk</wsse:BinarySecurityToken>

      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          <SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <Reference URI="cid:encsignexample">
            <Transforms>
              <Transform Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-
2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform" />
            </Transforms>
          </Reference>
        </SignedInfo>
      </Signature>
    </wsse:Security>
  </soap:Header>
  <!-- Body content -->

```


792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817

```
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>QTV...dw=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>H+x0...gUw=</SignatureValue>
<KeyInfo>
  <wsse:SecurityTokenReference>
    <wsse:Reference URI="#mySigCert" />
  </wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
</wsse:Security>
</soap:Header>
<soap:Body>
  <Ping xmlns="http://xmlsoap.org/Ping">
    <text>Acme Corp. - Scenario #3</text>
  </Ping>
</soap:Body>
</soap:Envelope>
--enc-sig-example
Content-Type: application/octet-stream
Content-Id: <encsignexample>
Content-Transfer-Encoding: base64
FEWMMIIIfc93ASjfdjsa358sa98xsjcx
```

818

819 5.6 Second Message - Response

820 5.6.1 Message Elements and Attributes

821 Items not listed in the following table MUST NOT be created or processed. Items marked
822 mandatory MUST be generated and processed. Items marked optional MAY be generated and
823 MUST be processed if present. Items MUST appear in the order specified, except as noted.

824

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

825

826 5.6.2 Message Creation

827 The response message MUST NOT contain a <wsse:Security> header. Any other header
828 elements MUST NOT be labeled with a mustUnderstand="1" attribute.

829 5.6.2.1 Security

830 There are no security properties on the response message

831 5.6.2.2 Body

832 The body element MUST be not be signed or encrypted

833 **5.6.3 Message Processing**

834 The response is passed to the application without modification.

835 **5.6.4 Example (Non-normative)**

836 Here is an example response.

```
837 <?xml version="1.0" encoding="utf-8" ?>
838 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
839 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
840 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
841 <soap:Body>
842 <PingResponse xmlns="http://xmlsoap.org/Ping">
843 <text> Acme Corp. - Scenario #3</text>
844 </PingResponse>
845 </soap:Body>
846 </soap:Envelope>
```

847

848 **5.7 Other processing**

849 This section describes processing that occurs outside of generating or processing a message.

850 **5.7.1 Requester**

851 No additional processing is required.

852 **5.7.2 Responder**

853 No additional processing is required.

854 **5.8 Expected Security Properties**

855 Use of the service is restricted to authorized parties that sign the attachment. The request
856 attachment is protected against modification and interception. The response is not protected in
857 any way.

858 **6 Scenario #4 – Attachment Signature and** 859 **Encryption with MIME Headers**

860 The SOAP request contains an attachment that has been signed and then encrypted. The
861 certificate associated with the encryption is provided out-of-band to the requestor. The certificate
862 used to verify the signature is provided in the header. The Response Body is not signed or
863 encrypted. There are two certificates in the request message. One identifies the recipient of the
864 encrypted attachment and one identifies the signer. This scenario differs from the first three
865 scenarios in that it covers MIME headers in the signature and encryption. This means that it uses
866 the Attachment-Complete Signature Reference Transform and Attachment-Complete
867 EncryptedData Type.

868 Aside from these two changes, this scenario is identical to Scenario #3.

869 **6.1 Attachment Properties**

870 This section specifies the attachment properties BEFORE security operations are applied. The
871 Content-Type of the attachment MUST be text/xml. The Content-Transfer-Encoding MUST be
872 8bit ASCII (8bit). The attachment MUST have a Content-Id header that uniquely identifies the
873 attachment. The generation of the Content-Id header is out of scope. An example of what the
874 attachment may look like before encryption and signature is shown as follows:

```
875 --enc-sig-headers-example  
876 Content-Type: text/xml; charset=UTF-8  
877 Content-Transfer-Encoding: 8bit  
878 Content-ID: <enc-sig-headers-example>  
879  
880 <?xml version=1.0" encoding="utf-8"?>  
881 <soxml/>
```

882 **6.2 Agreements**

883 This section describes the agreements that must be made, directly or indirectly between parties
884 who wish to interoperate.

885 **6.2.1 CERT-VALUE**

886 This is an opaque identifier indicating the X.509 certificate to be used. The certificate in question
887 MUST be obtained by the Requester by unspecified means. The certificate SHOULD NOT have a
888 KeyUsage extension. If it does contain a KeyUsage extension, it SHOULD include the values of
889 keyEncipherment, dataEncipherment and digitalSignature.

890 The Responder MUST have access to the private key corresponding to the public key in the
891 certificate.

892 **6.2.2 Signature Trust Root**

893 This refers generally to agreeing on at least one trusted key and any other certificates and
894 sources of revocation information sufficient to validate certificates sent for the purpose of
895 signature verification.

896 **6.3 Parameters**

897 This section describes parameters that are required to correctly create or process messages, but
898 not a matter of mutual agreement.

899 No parameters are required.

900 **6.4 General Message Flow**

901 This section provides a general overview of the flow of messages.

902 This contract covers a request/response MEP over the HTTP binding. SOAP 1.1 MUST be used.

903 The SOAP envelope MUST be wrapped in a SOAP Message Package as specified by **[SwA]**.

904 The Content-Transfer-Encoding for the encrypted attachment MUST be base64. As required by

905 SOAP 1.1, the SOAPAction HTTP header MUST be present. Any value, including a null string

906 may be used. The recipient SHOULD ignore the value. The request contains an attachment,

907 which is signed and then encrypted. The certificate for encryption is provided externally to the

908 requestor but conveyed in the request message. The attachment is encrypted with a random

909 symmetric key that is encrypted with a public key certificate. The certificate for signing is included

910 in the message. The Responder decrypts the attachment using its private key and then verifies

911 the signature using the included public key certificate. If no errors are detected it returns the

912 Response with no security properties.

913 **6.5 First Message - Request**

914 **6.5.1 Message Elements and Attributes**

915 Items not listed in the following table MAY be present, but MUST NOT be marked with the

916 mustUnderstand="1" attribute. Items marked mandatory MUST be generated and processed.

917 Items marked optional MAY be generated and MUST be processed if present. Items MUST

918 appear in the order specified, except as noted.

919

Name	Mandatory?
Envelope	Mandatory
Header	Mandatory
Security	Mandatory
mustUnderstand="1"	Mandatory
BinarySecurityToken	Mandatory
EncryptedKey	Mandatory
EncryptionMethod	Mandatory
KeyInfo	Mandatory
SecurityTokenReference	Mandatory
CipherData	Mandatory
ReferenceList	Mandatory
EncryptedData	Mandatory
EncryptionMethod	Mandatory
CipherData	Mandatory

CipherReference	Mandatory
Transforms	Mandatory
Transform	Mandatory
BinarySecurityToken	Mandatory
Signature	Mandatory
SignedInfo	Mandatory
CanonicalizationMethod	Mandatory
SignatureMethod	Mandatory
Reference	Mandatory
Transforms	Mandatory
Transform	Mandatory
SignatureValue	Mandatory
KeyInfo	Mandatory
Body	Mandatory
Ping	Mandatory

920 **6.5.2 Message Creation**

921 **6.5.2.1 Security**

922 The Security element MUST contain the mustUnderstand="1" attribute.

923 **6.5.2.2 BinarySecurityToken**

924 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
925 be labeled with an Id so it can be uniquely referenced. The value MUST be a PK certificate
926 suitable for encrypting the content. The certificate SHOULD NOT have a KeyUsage extension. If
927 it does contain a KeyUsage extension, it SHOULD include the value of keyEncipherment and
928 dataEncipherment.

929 **6.5.2.3 EncryptedKey**

930 The EncryptionMethod MUST contain the Algorithm attribute. The algorithm MUST be RSA v1.5.

931 The KeyInfo MUST contain a SecurityTokenReference with a Reference child that points to the
932 X.509 certificate of the recipient. The Reference child should point to a relative URI which
933 indicates the BinarySecurityToken containing the certificate which will be used to decrypt the
934 symmetric key.

935 The CipherData MUST contain the encrypted form of the random key, encrypted under the Public
936 Key specified in the specified X.509 certificate, using the specified algorithm.

937 The ReferenceList MUST contain a DataReference which has the value of a relative URI that
938 refers to the EncryptedData element that refers to the encrypted attachment.

939 **6.5.2.4 EncryptedData**

940 The EncryptedData element refers to the encrypted attachment. The Type attribute MUST be
941 present and it MUST have a value of #Attachment-Complete. The EncryptedData element MUST
942 be referenced by the ReferenceList element in the EncryptedKey element. The EncryptedData
943 MUST have a MimeType attribute with the value of text/xml.

944 **6.5.2.5 EncryptionMethod**

945 The encryption method MUST be Triple-DES in CBC mode.

946 **6.5.2.6 CipherData**

947 The CipherData MUST refer to the encrypted attachment with a CipherReference element. The
948 CipherReference element MUST refer to the attachment using a URI with a cid scheme. The
949 CipherReference must have a Transforms child with a single Transform sub child with the value
950 of #Attachment-Content-Only-Transform.

951 **6.5.2.7 BinarySecurityToken**

952 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
953 be labeled with an Id so it can be referenced by the signature. The value MUST be a PK
954 certificate suitable for verifying the signature. The certificate SHOULD NOT have a KeyUsage
955 extension. If it does contain a KeyUsage extension, it SHOULD include the values of
956 digitalSignature. The Requester must have access to the private key corresponding to the public
957 key in the certificate.

958 **6.5.2.8 Signature**

959 The signature is over the attachment content only, using the #Attachment-Content-Only-
960 Transform

961 **6.5.2.8.1 SignedInfo**

962 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST
963 be RSA-SHA1.

964 **6.5.2.8.2 Reference**

965 The Reference MUST specify a URI using the cid scheme that points to the Content-Id of the
966 attachment. The only Transform specified MUST be #Attachment-Content-Only. The
967 DigestMethod MUST be SHA1.

968 **6.5.2.8.3 SignatureValue**

969 The SignatureValue MUST be calculated as specified by the specification, using the private key
970 corresponding to the public key specified in the certificate in the BinarySecurityToken.

971 **6.5.2.8.4 KeyInfo**

972 The KeyInfo MUST contain a SecurityTokenReference with a reference to a relative URI which
973 indicates the BinarySecurityToken containing the certificate which will be used for signature
974 verification.

975 **6.5.2.9 Body**

976 The contents of the body MUST not be encrypted or signed

977 **6.5.2.10 Post Operation Attachment Properties**

978 This section specifies the attachment properties AFTER security operations are applied. The
979 Content-Type of the attachment MUST be application/octet-stream. The Content-Transfer-
980 Encoding MUST be base64. The attachment MUST have a Content-Id header that uniquely
981 identifies the attachment. The generation of the Content-Id header is out of scope. The Content-Id
982 MUST match the Content-Id before encryption.

983 **6.5.3 Responder Message Processing**

984 This section describes the processing performed by the Responder. If an error is detected, the
985 Responder MUST cease processing the message and issue a Fault with a value of
986 FailedAuthentication.

987 **6.5.3.1 Security**

988 **6.5.3.2 BinarySecurityToken**

989 The public key in the certificate MUST be used to decrypt the symmetric key. No trust validation
990 of the public key is required. The responder MUST have the matching private key.

991 **6.5.3.3 EncryptedKey**

992 The random key contained in the CipherData MUST be decrypted using the private key
993 corresponding to the certificate specified by the SecurityTokenReference, using the specified
994 algorithm.

995 **6.5.3.4 EncryptedData**

996 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted
997 symmetric key.

998 **6.5.3.5 Attachment**

999 After decrypting the attachment, it should have its signature verified

1000 **6.5.3.6 BinarySecurityToken**

1001 The certificate in the token MUST be validated. The Subject of the certificate MUST be an
1002 authorized entity. The public key in the certificate MUST be retained for verification of the
1003 signature.

1004 **6.5.3.7 Signature**

1005 The attachment MUST be verified against the signature using the specified algorithms and
1006 transforms and the retained public key.

1007 **6.5.3.8 Attachment**

1008 After the attachment's signature has been verified, it should be passed to the application

1009 **6.5.4 Example (Non-normative)**

1010 Here is an example request.

```
1011 Content-Type: multipart/related; boundary="enc-sig-headers-example";  
1012 type="text/xml"
```

```

1013 --enc-sig-headers-example
1014 Content-Type: text/xml; charset=utf-8
1015
1016 <?xml version="1.0" encoding="utf-8" ?>
1017 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
1018 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1019 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
1020   <soap:Header>
1021     <wsse:Security soap:mustUnderstand="1"
1022       xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
1023       secext-1.0.xsd">
1024
1025       <!-- This certificate is used for symmetric key encryption -->
1026       <wsse:BinarySecurityToken
1027         ValueType="wsse:X509v3"
1028         EncodingType="wsse:Base64Binary"
1029         xmlns:wsu="http://docs.oasis
1030         open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
1031         wsu:Id="myEncCert">MII...hk</wsse:BinarySecurityToken>
1032
1033       <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1034         <xenc:EncryptionMethod
1035           Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
1036         <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
1037           <wsse:SecurityTokenReference>
1038             <wsse:Reference URI="#myEncCert" />
1039           </wsse:SecurityTokenReference>
1040         </KeyInfo>
1041         <xenc:CipherData>
1042           <xenc:CipherValue>dNYS...fQ</xenc:CipherValue>
1043         </xenc:CipherData>
1044         <xenc:ReferenceList>
1045           <xenc:DataReference URI="#encrypted-signed-attachment" />
1046         </xenc:ReferenceList>
1047       </xenc:EncryptedKey>
1048
1049       <!-- The EncryptedData portion here refers to content of the attachment -->
1050
1051       <xenc:EncryptedData wsu:Id="encrypted-signed-attachment-headers"
1052         Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
1053         1.0#Attachment-Complete" MimeType="text/xml">
1054         xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1055         <xenc:EncryptionMethod
1056           Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
1057         <xenc:CipherData>
1058           <xenc:CipherReference URI="cid:encsign-headers-example">
1059             <xenc:Transforms>
1060               <ds:Transform
1061                 Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
1062                 profile-1.0#Attachment-Content-Only-Transform" />
1063             </xenc:Transforms>
1064           </xenc:CipherReference>
1065         </xenc:CipherData>
1066       </xenc:EncryptedData>
1067
1068       <!-- This certificate is used to verify the signature -->
1069       <wsse:BinarySecurityToken ValueType="wsse:X509v3"
1070         EncodingType="wsse:Base64Binary" xmlns:wsu="http://docs.oasis
1071         open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
1072         wsu:Id="mySigCert">MII...hk</wsse:BinarySecurityToken>
1073
1074       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
1075         <SignedInfo>
1076           <CanonicalizationMethod
1077             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
1078           <SignatureMethod
1079             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
1080           <Reference URI="cid:encsign-headers-example">
1081             <Transforms>

```


1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110

```
<Transform Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete-Transform"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmlsig#sha1"/>
<DigestValue>QTV...dw=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>H+x0...gUw=</SignatureValue>
<KeyInfo>
  <wsse:SecurityTokenReference>
    <wsse:Reference URI="#mySigCert" />
  </wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
</wsse:Security>
</soap:Header>
<soap:Body>
  <Ping xmlns="http://xmlsoap.org/Ping">
    <text>Acme Corp. - Scenario #4</text>
  </Ping>
</soap:Body>
</soap:Envelope>
--enc-sig-headers-example
Content-Type: application/octet-stream
Content-Id: <encsign-headers-example>
Content-Transfer-Encoding: base64
MW4dsa59fdsaSDr5hjdsKxhMW4dsa59ffds
```

6.6 Second Message - Response

6.6.1 Message Elements and Attributes

Items not listed in the following table MUST NOT be created or processed. Items marked mandatory MUST be generated and processed. Items marked optional MAY be generated and MUST be processed if present. Items MUST appear in the order specified, except as noted.

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

6.6.2 Message Creation

The response message MUST NOT contain a <wsse:Security> header. Any other header elements MUST NOT be labeled with a mustUnderstand="1" attribute.

6.6.2.1 Security

There are no security properties on the response message

6.6.2.2 Body

The body element MUST be not be signed or encrypted

1125 **6.6.3 Message Processing**

1126 The response is passed to the application without modification.

1127 **6.6.4 Example (Non-normative)**

1128 Here is an example response.

```
1129 <?xml version="1.0" encoding="utf-8" ?>
1130 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
1131 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1132 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
1133 <soap:Body>
1134 <PingResponse xmlns="http://xmlsoap.org/Ping">
1135 <text> Acme Corp. - Scenario #4</text>
1136 </PingResponse>
1137 </soap:Body>
1138 </soap:Envelope>
```

1139

1140 **6.7 Other processing**

1141 This section describes processing that occurs outside of generating or processing a message.

1142 **6.7.1 Requester**

1143 No additional processing is required.

1144 **6.7.2 Responder**

1145 No additional processing is required.

1146 **6.8 Expected Security Properties**

1147 Use of the service is restricted to authorized parties that sign the attachment. The request
1148 attachment is protected against modification and interception. The response is not protected in
1149 any way.

1150

1151 **7 References**

1152 **7.1 Normative**

- 1153 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
1154 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 1155 **[SwA]** W3C Note, "SOAP Messages with Attachments", 11 December 2000,
1156 <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-2001211>.
- 1157 **[WSS-SwA]** Hirsch, Frederick, *Web Services Security SOAP Message with Attachments*
1158 Profile 1.0, OASIS Draft 12 2004

Appendix A. Ping Application WSDL File

```

1160 <definitions xmlns:tns="http://xmlsoap.org/Ping" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1161 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
1162 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"
1163 targetNamespace="http://xmlsoap.org/Ping" name="Ping">
1164 <types>
1165 <schema targetNamespace="http://xmlsoap.org/Ping" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
1166 <import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd" schemaLocation="http://docs.oasis-
1167 open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"/>
1168
1169 <element name="text" type="xsd:string" nillable="true"/>
1170 <complexType name="ping">
1171 <sequence>
1172 <element ref="tns:text"/>
1173 </sequence>
1174 </complexType>
1175 <complexType name="pingResponse">
1176 <sequence>
1177 <element ref="tns:text"/>
1178 </sequence>
1179 </complexType>
1180 <element name="Ping" type="tns:ping"/>
1181 <element name="PingResponse" type="tns:pingResponse"/>
1182 </schema>
1183 </types>
1184 <message name="PingRequest">
1185 <part name="ping" element="tns:ping"/>
1186 </message>
1187 <message name="PingResponse">
1188 <part name="pingResponse" element="tns:PingResponse"/>
1189 </message>
1190 <portType name="PingPort">
1191 <operation name="Ping">
1192 <input message="tns:PingRequest"/>
1193 <output message="tns:PingResponse"/>
1194 </operation>
1195 </portType>
1196 <binding name="PingBinding" type="tns:PingPort">
1197 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1198 <operation name="Ping">
1199 <soap:operation/>
1200 <input>
1201 <mime:multipartRelated>
1202 <mime:part>
1203 <soap:body use="literal"/>
1204 </mime:part>
1205 <mime:part>
1206 <mime:content type="*/"/>
1207 </mime:part>
1208 </mime:multipartRelated>
1209 </input>
1210 <output>
1211 <soap:body use="literal"/>
1212 </output>
1213 </operation>
1214 </binding>
1215 <service name="PingService">
1216 <port name="Ping1" binding="tns:PingBinding">
1217 <soap:address location="http://localhost:9080/pingservice/Ping1"/>
1218 </port>
1219 <port name="Ping2" binding="tns:PingBinding">
1220 <soap:address location="http://localhost:9080/pingservice/Ping2"/>
1221 </port>
1222 <port name="Ping3" binding="tns:PingBinding">
1223 <soap:address location="http://localhost:9080/pingservice/Ping3"/>
1224 </port>
1225 <port name="Ping4" binding="tns:PingBinding">
1226 <soap:address location="http://localhost:9080/pingservice/Ping4"/>
1227 </port>
1228 <port name="Ping5" binding="tns:PingBinding">
1229 <soap:address location="http://localhost:9080/pingservice/Ping5"/>
1230 </port>
1231 <port name="Ping6" binding="tns:PingBinding">
1232 <soap:address location="http://localhost:9080/pingservice/Ping6"/>
1233 </port>
1234 <port name="Ping7" binding="tns:PingBinding">
1235 <soap:address location="http://localhost:9080/pingservice/Ping7"/>
1236 </port>

```

1237 </service>
1238 </definitions>

1239

Appendix B. - Revision History

1240

Rev	Date	By Whom	What
01	2004-09-07	Blake Dournaee	Initial version
02	2004-10-18	Blake Dournaee	Fixed problems with examples, specifically the quoting in the MIME headers
03	2004-10-21	Blake Dournaee	Fixed issues with examples. Pushed base64 encoding to MIME layer and removed it as a transform. Added scenario #4.
04	2004-10-22	Blake Dournaee	Fixed more problems with the examples. Clarified the meaning of the shorthand URI notation

1241

1242

Appendix C. Notices

1243 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1244 that might be claimed to pertain to the implementation or use of the technology described in this
1245 document or the extent to which any license under such rights might or might not be available;
1246 neither does it represent that it has made any effort to identify any such rights. Information on
1247 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1248 website. Copies of claims of rights made available for publication and any assurances of licenses
1249 to be made available, or the result of an attempt made to obtain a general license or permission
1250 for the use of such proprietary rights by implementors or users of this specification, can be
1251 obtained from the OASIS Executive Director.

1252 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1253 applications, or other proprietary rights which may cover technology that may be required to
1254 implement this specification. Please address the information to the OASIS Executive Director.

1255 **Copyright © OASIS Open 2004. All Rights Reserved.**

1256 This document and translations of it may be copied and furnished to others, and derivative works
1257 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1258 published and distributed, in whole or in part, without restriction of any kind, provided that the
1259 above copyright notice and this paragraph are included on all such copies and derivative works.
1260 However, this document itself does not be modified in any way, such as by removing the
1261 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1262 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1263 Property Rights document must be followed, or as required to translate it into languages other
1264 than English.

1265 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1266 successors or assigns.

1267 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1268 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1269 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1270 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1271 PARTICULAR PURPOSE.