



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

Interop 1 Scenarios

Working Draft 06, 1 Nov 2004

Document identifier:

swa-interop1-draft-06.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>

Manveen Kaur, Sun <Manveen.Kaur@sun.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

28	Introduction	5
29	1.1 Terminology.....	5
30	2 Test Application	6
31	2.1 Example Ping Element.....	6
32	2.2 Example PingResponse Element.....	6
33	2.3 SOAP Message Packages.....	6
34	3 Scenario #1: Attachment Signature	8
35	3.1 Attachment Properties.....	8
36	3.2 Agreements	8
37	3.2.1 CERT-VALUE	8
38	3.2.2 Signature Trust Root.....	8
39	3.3 Parameters.....	8
40	3.4 General Message Flow	8
41	3.5 First Message – Request	9
42	3.5.1 Message Elements and Attributes	9
43	3.5.2 Message Creation.....	9
44	3.5.3 Responder Message Processing.....	10
45	3.5.4 Example (Non-normative).....	11
46	3.6 Second Message - Response	12
47	3.6.1 Message Elements and Attributes	12
48	3.6.2 Message Creation.....	12
49	3.6.3 Message Processing.....	12
50	3.6.4 Example (Non-normative).....	12
51	3.7 Other processing	12
52	3.7.1 Requester	12
53	3.7.2 Responder	13
54	3.8 Expected Security Properties.....	13
55	4 Scenario #2 – Attachment Encryption	14
56	4.1 Attachment Properties.....	14
57	4.2 Agreements	14
58	4.2.1 CERT-VALUE	14
59	4.2.2 Signature Trust Root.....	14
60	4.3 Parameters.....	14
61	4.4 General Message Flow	14
62	4.5 First Message - Request.....	15
63	4.5.1 Message Elements and Attributes	15
64	4.5.2 Message Creation.....	15
65	4.5.3 Responder Message Processing.....	17
66	4.5.4 Example (Non-normative).....	17
67	4.6 Second Message - Response	18

68	4.6.1 Message Elements and Attributes	18
69	4.6.2 Message Creation.....	19
70	4.6.3 Message Processing.....	19
71	4.6.4 Example (Non-normative)	19
72	4.7 Other processing	19
73	4.7.1 Requester	19
74	4.7.2 Responder	19
75	4.8 Expected Security Properties.....	19
76	5 Scenario #3 – Attachment Signature and Encryption.....	20
77	5.1 Attachment Properties.....	20
78	5.2 Agreements.....	20
79	5.2.1 CERT-VALUE	20
80	5.2.2 Signature Trust Root.....	20
81	5.3 Parameters.....	20
82	5.4 General Message Flow	21
83	5.5 First Message - Request.....	21
84	5.5.1 Message Elements and Attributes	21
85	5.5.2 Message Creation.....	22
86	5.5.3 Responder Message Processing.....	24
87	5.5.4 Example (Non-normative)	24
88	5.6 Second Message - Response	26
89	5.6.1 Message Elements and Attributes	26
90	5.6.2 Message Creation.....	26
91	5.6.3 Message Processing.....	27
92	5.6.4 Example (Non-normative)	27
93	5.7 Other processing	27
94	5.7.1 Requester	27
95	5.7.2 Responder	27
96	5.8 Expected Security Properties.....	27
97	6 Scenario #4 – Attachment Signature and Encryption with MIME Headers	28
98	6.1 Attachment Properties.....	28
99	6.2 Agreements.....	28
100	6.2.1 CERT-VALUE	28
101	6.2.2 Signature Trust Root.....	28
102	6.3 Parameters.....	28
103	6.4 General Message Flow	29
104	6.5 First Message - Request.....	29
105	6.5.1 Message Elements and Attributes	29
106	6.5.2 Message Creation.....	30
107	6.5.3 Responder Message Processing.....	32
108	6.5.4 Example (Non-normative)	32
109	6.6 Second Message - Response	34
110	6.6.1 Message Elements and Attributes	34

111	6.6.2 Message Creation.....	34
112	6.6.3 Message Processing.....	35
113	6.6.4 Example (Non-normative).....	35
114	6.7 Other processing.....	35
115	6.7.1 Requester	35
116	6.7.2 Responder	35
117	6.8 Expected Security Properties.....	35
118	7 References.....	36
119	7.1 Normative	36
120	Appendix A. Ping Application WSDL File	37
121	Appendix B. Revision History	39
122	Appendix C. Notices	40
123		

124 **Introduction**

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

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

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

140 **1.1 Terminology**

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

143 2 Test Application

144 All four scenarios use the same, simple application.

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

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

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

153 2.1 Example Ping Element

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

157 2.2 Example PingResponse Element

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

161 2.3 SOAP Message Packages

162 When SOAP attachments are used as specified in **[SwA]** each SOAP message is accompanied
163 by a MIME header and possibly multiple boundary parts. This is known as a SOAP message
164 package. All interoperability scenarios in this document assume that a proper SOAP message
165 package is constructed using the HTTP and MIME headers appropriate to **[SwA]**.

166 In particular, implementations should take care in distinguishing between the HTTP headers in
167 the SOAP message package and the start of the SOAP payload. For example, the following
168 Multipart/Related header belongs to the HTTP layer and not the main SOAP payload:

```
169 Content-Type: Multipart/Related; boundary=boundary1; type="text/xml"; start="<foo>"
```

170 The main SOAP payload begins with the first boundary. For example:

```
171 --boundary1  
172 Content-Type: text/xml; charset=utf-8  
173 Content-ID: <foo>  
174  
175 <?xml version='1.0' ?>  
176 <s11:Envelope xmlns:s11="http://schemas.xmlsoap.org/soap/envelope/" />
```

177 Interoperability of the SOAP message package format, including the appropriate use of the MIME
178 header and boundary semantics, is outside the scope of this interoperability document.

179 2.4 URI Shorthand Notation

180 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
---	--------------------------------

181

182 **3 Scenario #1: Attachment Signature**

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

186 **3.1 Attachment Properties**

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

191 **3.2 Agreements**

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

194 **3.2.1 CERT-VALUE**

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

199 **3.2.2 Signature Trust Root**

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

203 **3.3 Parameters**

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

206 No parameters are required.

207 **3.4 General Message Flow**

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

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

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

216 3.5 First Message – Request

217 3.5.1 Message Elements and Attributes

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

222

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

223 3.5.2 Message Creation

224 3.5.2.1 Security

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

226 3.5.2.2 BinarySecurityToken

227 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST
228 be labeled with an Id so it can be referenced by the signature. The value MUST be a public key
229 certificate suitable for verifying the signature. The certificate SHOULD NOT have a KeyUsage
230 extension. If it does contain a KeyUsage extension, it SHOULD include the value of
231 digitalSignature. The Requester must have access to the private key corresponding to the public
232 key in the certificate.

233 **3.5.2.3 Signature**

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

236 **3.5.2.3.1 SignedInfo**

237 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST
238 be RSA-SHA1.

239 **3.5.2.3.2 Reference**

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

243 **3.5.2.3.3 SignatureValue**

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

246 **3.5.2.3.4 KeyInfo**

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

250 **3.5.2.4 Post Operation Attachment Properties**

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

255 **3.5.3 Responder Message Processing**

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

259 **3.5.3.1 Security**

260 **3.5.3.2 BinarySecurityToken**

261 The certificate in the token MUST be validated. The Subject of the certificate MUST be an
262 authorized entity. The public key in the certificate MUST be retained for verification of the
263 signature.

264 **3.5.3.3 Signature**

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

267 **3.5.3.4 Attachment**

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

269

3.5.4 Example (Non-normative)

270

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

271

```
--sig-example
```

272

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

273

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

274

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

275

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

276

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

277

```
  <soap:Header>
```

278

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

279

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

280

281

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

282

```
        <wsse:BinarySecurityToken ValueType="http://docs.oasis-
```

283

```
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
```

284

```
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
```

285

```
message-security-1.0#Base64Binary" xmlns:wsu="http://docs.oasis
```

286

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

287

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

288

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

289

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

290

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

291

```
      <SignedInfo>
```

292

```
        <CanonicalizationMethod
```

293

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

294

```
        <SignatureMethod
```

295

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

296

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

297

```
          <Transforms>
```

298

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

299

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

300

```
          </Transforms>
```

301

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

302

```
          <DigestValue>QIV...dw</DigestValue>
```

303

```
        </Reference>
```

304

```
      </SignedInfo>
```

305

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

306

```
    </KeyInfo>
```

307

```
    <wsse:SecurityTokenReference>
```

308

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

309

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

310

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

311

```
  </Signature>
```

312

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

313

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

314

```
<soap:Body>
```

315

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

316

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

317

```
  </Ping>
```

318

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

319

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

320

```
--sig-example
```

321

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

322

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

323

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

324

```
Dcg3AdGFcFs3764fddSArk
```

325

326

327

328

329 3.6 Second Message - Response

330 3.6.1 Message Elements and Attributes

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

334

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

335

336 3.6.2 Message Creation

337 3.6.2.1 Security

338 There are no security properties on the response message

339 3.6.2.2 Body

340 The body element MUST be not be signed or encrypted

341 3.6.3 Message Processing

342 The response is passed to the application without modification.

343 3.6.4 Example (Non-normative)

344 Here is an example response.

```
345 <?xml version="1.0" encoding="utf-8" ?>  
346 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
347 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
348 xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
349 <soap:Body>  
350 <PingResponse xmlns="http://xmlsoap.org/Ping">  
351 <text> Acme Corp. - Scenario #1</text>  
352 </PingResponse>  
353 </soap:Body>  
354 </soap:Envelope>
```

355

356 3.7 Other processing

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

358 3.7.1 Requester

359 No additional processing is required.

360 **3.7.2 Responder**

361 No additional processing is required.

362 **3.8 Expected Security Properties**

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

366 4 Scenario #2 – Attachment Encryption

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

371 4.1 Attachment Properties

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

376 4.2 Agreements

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

379 4.2.1 CERT-VALUE

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

384 The Responder MUST have access to the Private key corresponding to the Public key in the
385 certificate.

386 4.2.2 Signature Trust Root

387 There is no digital signature operation for this scenario

388 4.3 Parameters

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

391 No parameters are required.

392 4.4 General Message Flow

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

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

404 4.5 First Message - Request

405 4.5.1 Message Elements and Attributes

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

410

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

411 4.5.2 Message Creation

412 4.5.2.1 Security

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

414 **4.5.2.2 BinarySecurityToken**

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

421 **4.5.2.3 EncryptedKey**

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

423 The KeyInfo MUST contain a SecurityTokenReference with a reference to a relative URI which
424 indicates the BinarySecurityToken containing the certificate which will be used to decrypt the
425 symmetric key.

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

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

430 **4.5.2.4 EncryptedData**

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

435 **4.5.2.5 EncryptionMethod**

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

437 **4.5.2.6 CipherData**

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

442

443 **4.5.2.7 Body**

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

445 **4.5.2.8 Ping**

446 The Ping element should contain the scenario number and the name of the entity performing the
447 request.

448 **4.5.2.9 Post Operation Attachment Properties**

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

454 **4.5.3 Responder Message Processing**

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

458 **4.5.3.1 Security**

459 **4.5.3.2 BinarySecurityToken**

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

462 **4.5.3.3 EncryptedKey**

463 The random key contained in the CipherData MUST be decrypted using the private key
464 corresponding to the certificate specified by the SecurityTokenReference, using the specified
465 algorithm.

466 **4.5.3.4 EncryptedData**

467 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted
468 symmetric key.

469 **4.5.3.5 Attachment**

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

471 **4.5.4 Example (Non-normative)**

472 Here is an example request.

```
473 Content-Type: multipart/related; boundary="enc-example"; type="text/xml"
474 --enc-example
475 Content-Type: text/xml; charset=utf-8
476
477 <?xml version="1.0" encoding="utf-8" ?>
478 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
479 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
480 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
481 <soap:Header>
482 <wsse:Security soap:mustUnderstand="1"
483 xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
484 secext-1.0.xsd">
485
486 <!-- This certificate is used for symmetric key encryption -->
487 <wsse:BinarySecurityToken
488 Value="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
489 token-profile-1.0#X509v3"
490 EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
491 message-security-1.0#Base64Binary"
492 xmlns:wsu="http://docs.oasis
493 open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
494 wsu:Id="myEncCert">MII...hk</wsse:BinarySecurityToken>
495
496 <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
497 <xenc:EncryptionMethod
498 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
499 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
500 <wsse:SecurityTokenReference>
501 <wsse:Reference URI="#myEncCert" />
502 </wsse:SecurityTokenReference>
503 </KeyInfo>
```

```

504     <xenc:CipherData>
505       <xenc:CipherValue>dNYS...fQ=</xenc:CipherValue>
506     </xenc:CipherData>
507     <xenc:ReferenceList>
508       <xenc:DataReference URI="#encrypted-attachment" />
509     </xenc:ReferenceList>
510   </xenc:EncryptedKey>
511
512   <!-- The EncryptedData portion here refers to content of the attachment -->
513
514   <xenc:EncryptedData Id="encrypted-attachment"
515     Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
516 1.0#Attachment-Content-Only" MimeType="image/jpeg">
517     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
518     <xenc:EncryptionMethod
519       Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
520     <xenc:CipherData>
521       <xenc:CipherReference URI="cid:enc">
522         <xenc:Transforms>
523           <ds:Transform
524             Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
525 profile-1.0#Attachment-Content-Only-Transform" />
526           <ds:Transform
527             />
528         </xenc:Transforms>
529       </xenc:CipherReference>
530     </xenc:CipherData>
531   </xenc:EncryptedData>
532
533 </wsse:Security>
534 </soap:Header>
535 <soap:Body>
536   <Ping xmlns="http://xmlsoap.org/Ping">
537     <text>Acme Corp. - Scenario #2</text>
538   </Ping>
539 </soap:Body>
540 </soap:Envelope>
541 --enc-example
542 Content-Type: application/octet-stream
543 Content-Id: <enc>
544 Content-Transfer-Encoding: base64
545 Dsh5SA3thsRh3Dh54wafDhjaq2

```

546

547 4.6 Second Message - Response

548 4.6.1 Message Elements and Attributes

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

552

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

553

554 **4.6.2 Message Creation**

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

557 **4.6.2.1 Security**

558 There are no security properties on the response message

559 **4.6.2.2 Body**

560 The body element MUST be not be signed or encrypted

561 **4.6.3 Message Processing**

562 The response is passed to the application without modification.

563 **4.6.4 Example (Non-normative)**

564 Here is an example response.

```
565 <?xml version="1.0" encoding="utf-8" ?>  
566 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
567 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
568 xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
569 <soap:Body>  
570 <PingResponse xmlns="http://xmlsoap.org/Ping">  
571 <text> Acme Corp. - Scenario #2</text>  
572 </PingResponse>  
573 </soap:Body>  
574 </soap:Envelope>
```

575 **4.7 Other processing**

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

577 **4.7.1 Requester**

578 No additional processing is required.

579 **4.7.2 Responder**

580 No additional processing is required.

581 **4.8 Expected Security Properties**

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

585 **5 Scenario #3 – Attachment Signature and**
586 **Encryption**

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

592 **5.1 Attachment Properties**

593 This section specifies the attachment properties BEFORE security operations are applied. The
594 Content-Type of the attachment MUST be text/xml. The Content-Transfer-Encoding MUST be
595 8bit ASCII (8bit). The attachment MUST have a Content-Id header that uniquely identifies the
596 attachment. The generation of the Content-Id header is out of scope. An example of what the
597 attachment may look like before encryption and signing is shown as follows. This example is non-
598 normative.

```
599 --enc-sig-example  
600 Content-Type: text/xml; charset=utf-8  
601 Content-Transfer-Encoding: 8bit  
602 Content-ID: <encsignexample>  
603  
604 <?xml version=1.0" encoding="utf-8"?>  
605 <somexml/>
```

606

607 **5.2 Agreements**

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

610 **5.2.1 CERT-VALUE**

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

615 The Responder MUST have access to the private key corresponding to the public key in the
616 certificate.

617 **5.2.2 Signature Trust Root**

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

621 **5.3 Parameters**

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

624 No parameters are required.

625 5.4 General Message Flow

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

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

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

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

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

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

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

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

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

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

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

637 Response with no security properties.

638 5.5 First Message - Request

639 5.5.1 Message Elements and Attributes

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

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

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

643 appear in the order specified, except as noted.

644

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

645 **5.5.2 Message Creation**

646 **5.5.2.1 Security**

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

648 **5.5.2.2 BinarySecurityToken**

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

654 **5.5.2.3 EncryptedKey**

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

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

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

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

664 **5.5.2.4 EncryptedData**

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

669 **5.5.2.5 EncryptionMethod**

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

671 **5.5.2.6 CipherData**

672 The CipherData MUST refer to the encrypted attachment with a CipherReference element. The
673 CipherReference element MUST refer to the attachment using a URI with a cid scheme. The
674 CipherReference must have a Transforms child with a single Transform sub child with the value
675 of #Attachment-Content-Only-Transform.

676 **5.5.2.7 BinarySecurityToken**

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

683 **5.5.2.8 Signature**

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

686 **5.5.2.8.1 SignedInfo**

687 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST
688 be RSA-SHA1.

689 **5.5.2.8.2 Reference**

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

693 **5.5.2.8.3 SignatureValue**

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

696 **5.5.2.8.4 KeyInfo**

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

700 **5.5.2.9 Body**

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

702 **5.5.2.10 Post Operation Attachment Properties**

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

708 **5.5.3 Responder Message Processing**

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

712 **5.5.3.1 Security**

713 **5.5.3.2 BinarySecurityToken**

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

716 **5.5.3.3 EncryptedKey**

717 The random key contained in the CipherData MUST be decrypted using the private key
718 corresponding to the certificate specified by the SecurityTokenReference, using the specified
719 algorithm.

720 **5.5.3.4 EncryptedData**

721 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted
722 symmetric key.

723 **5.5.3.5 Attachment**

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

725 **5.5.3.6 BinarySecurityToken**

726 The certificate in the token MUST be validated. The Subject of the certificate MUST be an
727 authorized entity. The public key in the certificate MUST be retained for verification of the
728 signature.

729 **5.5.3.7 Signature**

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

732 **5.5.3.8 Attachment**

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

734 **5.5.4 Example (Non-normative)**

735 Here is an example request.

```
736 Content-Type: multipart/related; boundary="enc-sig-example"; type="text/xml"  
737 --enc-sig-example  
738 Content-Type: text/xml; charset=utf-8
```


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
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807

```
<?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="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509v3"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#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 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="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
message-security-1.0#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"/>
        </SignedInfo>
      </Signature>
    </wsse:Security>
  </soap:Header>
  <!-- Body content -->

```

808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837

```
<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>
  <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
FEWMMIIfc93ASjfdjsa358sa98xsjcx
```

838

5.6 Second Message - Response

839

5.6.1 Message Elements and Attributes

840

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

844

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

845

5.6.2 Message Creation

846

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

5.6.2.1 Security

849

850 There are no security properties on the response message

851 **5.6.2.2 Body**

852 The body element MUST be not be signed or encrypted

853 **5.6.3 Message Processing**

854 The response is passed to the application without modification.

855 **5.6.4 Example (Non-normative)**

856 Here is an example response.

```
857 <?xml version="1.0" encoding="utf-8" ?>
858 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
859 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
860 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
861 <soap:Body>
862 <PingResponse xmlns="http://xmlsoap.org/Ping">
863 <text> Acme Corp. - Scenario #3</text>
864 </PingResponse>
865 </soap:Body>
866 </soap:Envelope>
```

867

868 **5.7 Other processing**

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

870 **5.7.1 Requester**

871 No additional processing is required.

872 **5.7.2 Responder**

873 No additional processing is required.

874 **5.8 Expected Security Properties**

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

878 **6 Scenario #4 – Attachment Signature and** 879 **Encryption with MIME Headers**

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

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

889 **6.1 Attachment Properties**

890 This section specifies the attachment properties BEFORE security operations are applied. The
891 Content-Type of the attachment MUST be text/xml. The Content-Transfer-Encoding MUST be
892 8bit ASCII (8bit). The attachment MUST have a Content-Id header that uniquely identifies the
893 attachment. The generation of the Content-Id header is out of scope. An example of what the
894 attachment may look like before encryption and signature is shown as follows:

```
895 --enc-sig-headers-example  
896 Content-Type: text/xml; charset=UTF-8  
897 Content-Transfer-Encoding: 8bit  
898 Content-ID: <enc-sig-headers-example>  
899  
900 <?xml version=1.0" encoding="utf-8"?>  
901 <soxml/>
```

902 **6.2 Agreements**

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

905 **6.2.1 CERT-VALUE**

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

910 The Responder MUST have access to the private key corresponding to the public key in the
911 certificate.

912 **6.2.2 Signature Trust Root**

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

916 **6.3 Parameters**

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

919 No parameters are required.

920 **6.4 General Message Flow**

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

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

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

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

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

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

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

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

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

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

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

932 Response with no security properties.

933 **6.5 First Message - Request**

934 **6.5.1 Message Elements and Attributes**

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

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

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

938 appear in the order specified, except as noted.

939

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

940 **6.5.2 Message Creation**

941 **6.5.2.1 Security**

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

943 **6.5.2.2 BinarySecurityToken**

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

949 **6.5.2.3 EncryptedKey**

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

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

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

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

959 **6.5.2.4 EncryptedData**

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

964 **6.5.2.5 EncryptionMethod**

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

966 **6.5.2.6 CipherData**

967 The CipherData MUST refer to the encrypted attachment with a CipherReference element. The
968 CipherReference element MUST refer to the attachment using a URI with a cid scheme. The
969 CipherReference must have a Transforms child with a single Transform sub child with the value
970 of #Attachment-Content-Only-Transform.

971 **6.5.2.7 BinarySecurityToken**

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

978 **6.5.2.8 Signature**

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

981 **6.5.2.8.1 SignedInfo**

982 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST
983 be RSA-SHA1.

984 **6.5.2.8.2 Reference**

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

988 **6.5.2.8.3 SignatureValue**

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

991 **6.5.2.8.4 KeyInfo**

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

995 **6.5.2.9 Body**

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

997 **6.5.2.10 Post Operation Attachment Properties**

998 This section specifies the attachment properties AFTER security operations are applied. The
999 Content-Type of the attachment MUST be application/octet-stream. The Content-Transfer-
1000 Encoding MUST be base64. The attachment MUST have a Content-Id header that uniquely
1001 identifies the attachment. The generation of the Content-Id header is out of scope. The Content-Id
1002 MUST match the Content-Id before encryption.

1003 **6.5.3 Responder Message Processing**

1004 This section describes the processing performed by the Responder. If an error is detected, the
1005 Responder MUST cease processing the message and issue a Fault with a value of
1006 FailedAuthentication.

1007 **6.5.3.1 Security**

1008 **6.5.3.2 BinarySecurityToken**

1009 The public key in the certificate MUST be used to decrypt the symmetric key. No trust validation
1010 of the public key is required. The responder MUST have the matching private key.

1011 **6.5.3.3 EncryptedKey**

1012 The random key contained in the CipherData MUST be decrypted using the private key
1013 corresponding to the certificate specified by the SecurityTokenReference, using the specified
1014 algorithm.

1015 **6.5.3.4 EncryptedData**

1016 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted
1017 symmetric key.

1018 **6.5.3.5 Attachment**

1019 After decrypting the attachment, it should have its signature verified

1020 **6.5.3.6 BinarySecurityToken**

1021 The certificate in the token MUST be validated. The Subject of the certificate MUST be an
1022 authorized entity. The public key in the certificate MUST be retained for verification of the
1023 signature.

1024 **6.5.3.7 Signature**

1025 The attachment MUST be verified against the signature using the specified algorithms and
1026 transforms and the retained public key.

1027 **6.5.3.8 Attachment**

1028 After the attachment's signature has been verified, it should be passed to the application

1029 **6.5.4 Example (Non-normative)**

1030 Here is an example request.

```
1031 Content-Type: multipart/related; boundary="enc-sig-headers-example";  
1032 type="text/xml"
```



```

1033 --enc-sig-headers-example
1034 Content-Type: text/xml; charset=utf-8
1035
1036 <?xml version="1.0" encoding="utf-8" ?>
1037 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
1038 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1039 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
1040   <soap:Header>
1041     <wsse:Security soap:mustUnderstand="1"
1042       xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
1043       secext-1.0.xsd">
1044
1045       <!-- This certificate is used for symmetric key encryption -->
1046       <wsse:BinarySecurityToken
1047         ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
1048         token-profile-1.0#X509v3"
1049         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
1050         message-security-1.0#Base64Binary"
1051         xmlns:wsu="http://docs.oasis
1052         open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
1053         wsu:Id="myEncCert">MII...hk</wsse:BinarySecurityToken>
1054
1055       <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1056         <xenc:EncryptionMethod
1057           Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
1058         <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
1059           <wsse:SecurityTokenReference>
1060             <wsse:Reference URI="#myEncCert" />
1061           </wsse:SecurityTokenReference>
1062         </KeyInfo>
1063         <xenc:CipherData>
1064           <xenc:CipherValue>dNYS...fQ=</xenc:CipherValue>
1065         </xenc:CipherData>
1066         <xenc:ReferenceList>
1067           <xenc:DataReference URI="#encrypted-signed-attachment" />
1068         </xenc:ReferenceList>
1069       </xenc:EncryptedKey>
1070
1071       <!-- The EncryptedData portion here refers to content of the attachment -->
1072
1073       <xenc:EncryptedData Id="encrypted-signed-attachment-headers"
1074         Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
1075         1.0#Attachment-Complete" MimeType="text/xml">
1076         xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
1077         <xenc:EncryptionMethod
1078           Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
1079         <xenc:CipherData>
1080           <xenc:CipherReference URI="cid:encsign-headers-example">
1081             <xenc:Transforms>
1082               <ds:Transform
1083                 Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
1084                 profile-1.0#Attachment-Content-Only-Transform" />
1085             </xenc:Transforms>
1086           </xenc:CipherReference>
1087         </xenc:CipherData>
1088       </xenc:EncryptedData>
1089
1090       <!-- This certificate is used to verify the signature -->
1091       <wsse:BinarySecurityToken ValueType="http://docs.oasis-
1092       open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
1093       EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
1094       message-security-1.0#Base64Binary" xmlns:wsu="http://docs.oasis
1095       open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
1096       wsu:Id="mySigCert">MII...hk</wsse:BinarySecurityToken>
1097
1098       <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
1099         <SignedInfo>
1100           <CanonicalizationMethod
1101             Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />

```

1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

```
<SignatureMethod
  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="cid:encsign-headers-example">
  <Transforms>
    <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/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 #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
```

1134

6.6 Second Message - Response

1135

6.6.1 Message Elements and Attributes

1136

1137 Items not listed in the following table MUST NOT be created or processed. Items marked
1138 mandatory MUST be generated and processed. Items marked optional MAY be generated and
1139 MUST be processed if present. Items MUST appear in the order specified, except as noted.

1140

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

1141

6.6.2 Message Creation

1142

1143 The response message MUST NOT contain a <wsse:Security> header. Any other header
1144 elements MUST NOT be labeled with a mustUnderstand="1" attribute.

6.6.2.1 Security

1145

1146 There are no security properties on the response message

1147 **6.6.2.2 Body**

1148 The body element MUST be not be signed or encrypted

1149 **6.6.3 Message Processing**

1150 The response is passed to the application without modification.

1151 **6.6.4 Example (Non-normative)**

1152 Here is an example response.

```
1153 <?xml version="1.0" encoding="utf-8" ?>
1154 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
1155 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1156 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
1157 <soap:Body>
1158 <PingResponse xmlns="http://xmlsoap.org/Ping">
1159 <text> Acme Corp. - Scenario #4</text>
1160 </PingResponse>
1161 </soap:Body>
1162 </soap:Envelope>
```

1163 **6.7 Other processing**

1164 This section describes processing that occurs outside of generating or processing a message.

1165 **6.7.1 Requester**

1166 No additional processing is required.

1167 **6.7.2 Responder**

1168 No additional processing is required.

1169 **6.8 Expected Security Properties**

1170 Use of the service is restricted to authorized parties that sign the attachment. The request
1171 attachment is protected against modification and interception. The response is not protected in
1172 any way.

1173

1174 **7 References**

1175 **7.1 Normative**

- 1176 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
1177 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 1178 **[SwA]** W3C Note, "SOAP Messages with Attachments", 11 December 2000,
1179 <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-2001211>.
- 1180 **[WSS-SwA]** Hirsch, Frederick, *Web Services Security SOAP Message with Attachments*
1181 *Profile 1.0*, OASIS Draft 12 2004

Appendix A. Ping Application WSDL File

```

1183 <definitions xmlns:tns="http://xmlsoap.org/Ping" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1184 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
1185 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"
1186 targetNamespace="http://xmlsoap.org/Ping" name="Ping">
1187 <types>
1188 <schema targetNamespace="http://xmlsoap.org/Ping" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
1189 <import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd" schemaLocation="http://docs.oasis-
1190 open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"/>
1191
1192 <element name="text" type="xsd:string" nillable="true"/>
1193 <complexType name="ping">
1194 <sequence>
1195 <element ref="tns:text"/>
1196 </sequence>
1197 </complexType>
1198 <complexType name="pingResponse">
1199 <sequence>
1200 <element ref="tns:text"/>
1201 </sequence>
1202 </complexType>
1203 <element name="Ping" type="tns:ping"/>
1204 <element name="PingResponse" type="tns:pingResponse"/>
1205 </schema>
1206 </types>
1207 <message name="PingRequest">
1208 <part name="ping" element="tns:ping"/>
1209 </message>
1210 <message name="PingResponse">
1211 <part name="pingResponse" element="tns:PingResponse"/>
1212 </message>
1213 <portType name="PingPort">
1214 <operation name="Ping">
1215 <input message="tns:PingRequest"/>
1216 <output message="tns:PingResponse"/>
1217 </operation>
1218 </portType>
1219 <binding name="PingBinding" type="tns:PingPort">
1220 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1221 <operation name="Ping">
1222 <soap:operation/>
1223 <input>
1224 <mime:multipartRelated>
1225 <mime:part>
1226 <soap:body use="literal"/>
1227 </mime:part>
1228 <mime:part>
1229 <mime:content type="*/"/>
1230 </mime:part>
1231 </mime:multipartRelated>
1232 </input>
1233 <output>
1234 <soap:body use="literal"/>
1235 </output>
1236 </operation>
1237 </binding>
1238 <service name="PingService">
1239 <port name="Ping1" binding="tns:PingBinding">
1240 <soap:address location="http://localhost:9080/pingservice/Ping1"/>
1241 </port>
1242 <port name="Ping2" binding="tns:PingBinding">
1243 <soap:address location="http://localhost:9080/pingservice/Ping2"/>
1244 </port>
1245 <port name="Ping3" binding="tns:PingBinding">
1246 <soap:address location="http://localhost:9080/pingservice/Ping3"/>
1247 </port>
1248 <port name="Ping4" binding="tns:PingBinding">
1249 <soap:address location="http://localhost:9080/pingservice/Ping4"/>
1250 </port>
1251 <port name="Ping5" binding="tns:PingBinding">
1252 <soap:address location="http://localhost:9080/pingservice/Ping5"/>
1253 </port>
1254 <port name="Ping6" binding="tns:PingBinding">
1255 <soap:address location="http://localhost:9080/pingservice/Ping6"/>
1256 </port>
1257 <port name="Ping7" binding="tns:PingBinding">
1258 <soap:address location="http://localhost:9080/pingservice/Ping7"/>
1259 </port>

```

1260 </service>
1261 </definitions>

1262

Appendix B. - Revision History

1263

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
05	2004-10-28	Blake Dournaee	Added fully qualified URIs to the examples for the X.509 token attributes ValueType and EncodingType. Added text to disambiguate between the HTTP headers and the SOAP payload.

1264

1265

Appendix C. Notices

1266 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1267 that might be claimed to pertain to the implementation or use of the technology described in this
1268 document or the extent to which any license under such rights might or might not be available;
1269 neither does it represent that it has made any effort to identify any such rights. Information on
1270 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1271 website. Copies of claims of rights made available for publication and any assurances of licenses
1272 to be made available, or the result of an attempt made to obtain a general license or permission
1273 for the use of such proprietary rights by implementors or users of this specification, can be
1274 obtained from the OASIS Executive Director.

1275 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1276 applications, or other proprietary rights which may cover technology that may be required to
1277 implement this specification. Please address the information to the OASIS Executive Director.

1278 **Copyright © OASIS Open 2004. All Rights Reserved.**

1279 This document and translations of it may be copied and furnished to others, and derivative works
1280 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1281 published and distributed, in whole or in part, without restriction of any kind, provided that the
1282 above copyright notice and this paragraph are included on all such copies and derivative works.
1283 However, this document itself does not be modified in any way, such as by removing the
1284 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1285 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1286 Property Rights document must be followed, or as required to translate it into languages other
1287 than English.

1288 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1289 successors or assigns.

1290 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1291 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1292 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1293 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1294 PARTICULAR PURPOSE.