



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24

---

# Web Services Security SOAP Messages with Attachments (SwA) Profile 1.0 Interop 1 Scenarios Working Draft 03, 21 Oct 2004

**Document identifier:**

swa-interop1-draft-03.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>

**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](mailto:wss@lists.oasis-open.org) list. Others should subscribe to and send comments to the [wss-comment@lists.oasis-open.org](mailto:wss-comment@lists.oasis-open.org) list. To subscribe, send an email message to [wss-comment-request@lists.oasis-open.org](mailto:wss-comment-request@lists.oasis-open.org) with the word "subscribe" as the body of the message.

## Table of Contents

26	Introduction .....	5
27	1.1 Terminology.....	5
28	2 Test Application .....	6
29	2.1 Example Ping Element.....	6
30	2.2 Example PingResponse Element.....	6
31	2.3 SOAP Message Packages.....	6
32	3 Scenario #1: Attachment Signature .....	7
33	3.1 Attachment Properties.....	7
34	3.2 Agreements .....	7
35	3.2.1 CERT-VALUE .....	7
36	3.2.2 Signature Trust Root.....	7
37	3.3 Parameters.....	7
38	3.4 General Message Flow .....	7
39	3.5 First Message – Request .....	8
40	3.5.1 Message Elements and Attributes .....	8
41	3.5.2 Message Creation.....	8
42	3.5.3 Responder Message Processing.....	9
43	3.5.4 Example (Non-normative).....	10
44	3.6 Second Message - Response .....	10
45	3.6.1 Message Elements and Attributes .....	10
46	3.6.2 Message Creation.....	11
47	3.6.3 Message Processing.....	11
48	3.6.4 Example (Non-normative).....	11
49	3.7 Other processing .....	11
50	3.7.1 Requester .....	11
51	3.7.2 Responder .....	11
52	3.8 Expected Security Properties.....	11
53	4 Scenario #2 – Attachment Encryption .....	12
54	4.1 Attachment Properties.....	12
55	4.2 Agreements .....	12
56	4.2.1 CERT-VALUE .....	12
57	4.2.2 Signature Trust Root.....	12
58	4.3 Parameters.....	12
59	4.4 General Message Flow .....	12
60	4.5 First Message - Request.....	13
61	4.5.1 Message Elements and Attributes .....	13
62	4.5.2 Message Creation.....	13
63	4.5.3 Responder Message Processing.....	15
64	4.5.4 Example (Non-normative).....	15
65	4.6 Second Message - Response .....	16
66	4.6.1 Message Elements and Attributes .....	16
67	4.6.2 Message Creation.....	16

68	4.6.3 Message Processing.....	17
69	4.6.4 Example (Non-normative).....	17
70	4.7 Other processing.....	17
71	4.7.1 Requester .....	17
72	4.7.2 Responder .....	17
73	4.8 Expected Security Properties.....	17
74	5 Scenario #3 – Attachment Signature and Encryption.....	18
75	5.1 Attachment Properties.....	18
76	5.2 Agreements.....	18
77	5.2.1 CERT-VALUE .....	18
78	5.2.2 Signature Trust Root.....	18
79	5.3 Parameters.....	18
80	5.4 General Message Flow .....	19
81	5.5 First Message - Request.....	19
82	5.5.1 Message Elements and Attributes .....	19
83	5.5.2 Message Creation.....	20
84	5.5.3 Responder Message Processing.....	22
85	5.5.4 Example (Non-normative).....	22
86	5.6 Second Message - Response .....	24
87	5.6.1 Message Elements and Attributes .....	24
88	5.6.2 Message Creation.....	24
89	5.6.3 Message Processing.....	24
90	5.6.4 Example (Non-normative).....	24
91	5.7 Other processing.....	25
92	5.7.1 Requester .....	25
93	5.7.2 Responder .....	25
94	5.8 Expected Security Properties.....	25
95	6 Scenario #4 – Attachment Signature and Encryption with MIME Headers .....	26
96	6.1 Attachment Properties.....	26
97	6.2 Agreements.....	26
98	6.2.1 CERT-VALUE .....	26
99	6.2.2 Signature Trust Root.....	26
100	6.3 Parameters.....	26
101	6.4 General Message Flow .....	27
102	6.5 First Message - Request.....	27
103	6.5.1 Message Elements and Attributes .....	27
104	6.5.2 Message Creation.....	28
105	6.5.3 Responder Message Processing.....	30
106	6.5.4 Example (Non-normative).....	30
107	6.6 Second Message - Response .....	32
108	6.6.1 Message Elements and Attributes .....	32
109	6.6.2 Message Creation.....	32
110	6.6.3 Message Processing.....	32
111	6.6.4 Example (Non-normative).....	32

112	6.7 Other processing.....	33
113	6.7.1 Requester .....	33
114	6.7.2 Responder .....	33
115	6.8 Expected Security Properties.....	33
116	7 References.....	34
117	7.1 Normative .....	34
118	Appendix A. Ping Application WSDL File.....	35
119	Appendix B. Revision History .....	36
120	Appendix C. Notices .....	37
121		

---

## 122 Introduction

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

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

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

### 138 1.1 Terminology

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

---

## 141 2 Test Application

142 All four scenarios use the same, simple application.

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

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

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

### 151 2.1 Example Ping Element

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

### 155 2.2 Example PingResponse Element

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

### 159 2.3 SOAP Message Packages

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

---

## 166 **3 Scenario #1: Attachment Signature**

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

### 170 **3.1 Attachment Properties**

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

### 175 **3.2 Agreements**

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

#### 178 **3.2.1 CERT-VALUE**

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

#### 183 **3.2.2 Signature Trust Root**

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

### 187 **3.3 Parameters**

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

190 No parameters are required.

### 191 **3.4 General Message Flow**

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

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

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

## 200 3.5 First Message – Request

### 201 3.5.1 Message Elements and Attributes

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

206

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

### 207 3.5.2 Message Creation

#### 208 3.5.2.1 Security

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

#### 210 3.5.2.2 BinarySecurityToken

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



### 217 **3.5.2.3 Signature**

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

#### 220 **3.5.2.3.1 SignedInfo**

221 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST  
222 be RSA-SHA1.

#### 223 **3.5.2.3.2 Reference**

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

#### 227 **3.5.2.3.3 SignatureValue**

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

#### 230 **3.5.2.3.4 KeyInfo**

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

### 234 **3.5.2.4 Post Operation Attachment Properties**

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

## 239 **3.5.3 Responder Message Processing**

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

### 243 **3.5.3.1 Security**

#### 244 **3.5.3.2 BinarySecurityToken**

245 The certificate in the token MUST be validated. The Subject of the certificate MUST be an  
246 authorized entity. The public key in the certificate MUST be retained for verification of the  
247 signature.

#### 248 **3.5.3.3 Signature**

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

#### 251 **3.5.3.4 Attachment**

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

253

## 3.5.4 Example (Non-normative)

254

```
Content-Type: multipart/related; boundary="sig-example"; type="text/xml"
--sig-example
256 Content-Type: text/xml; charset=utf-8
257
258 <?xml version="1.0" encoding="utf-8" ?>
259 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
260   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
261   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
262   <soap:Header>
263     <wsse:Security soap:mustUnderstand="1"
264       xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
265       secext-1.0.xsd">
266
267       <!-- This is the certificate used to verify the signature -->
268       <wsse:BinarySecurityToken ValueType="wsse:X509v3"
269         EncodingType="wsse:Base64Binary" xmlns:wsu="http://docs.oasis
270         open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
271         wsu:Id="mySigCert">MII...hk</wsse:BinarySecurityToken>
272
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
304 --sig-example
305 Content-Type: image/jpeg
306 Content-Id: <signature>
307 Content-Transfer-Encoding: base64
308
309 Dcg3AdGFcFs3764fddSArk
```

310

## 311 3.6 Second Message - Response

### 312 3.6.1 Message Elements and Attributes

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

316

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

317

## 318 **3.6.2 Message Creation**

### 319 **3.6.2.1 Security**

320 There are no security properties on the response message

### 321 **3.6.2.2 Body**

322 The body element MUST be not be signed or encrypted

## 323 **3.6.3 Message Processing**

324 The response is passed to the application without modification.

## 325 **3.6.4 Example (Non-normative)**

326 Here is an example response.

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

337

## 338 **3.7 Other processing**

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

### 340 **3.7.1 Requester**

341 No additional processing is required.

### 342 **3.7.2 Responder**

343 No additional processing is required.

## 344 **3.8 Expected Security Properties**

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

---

## 348 **4 Scenario #2 – Attachment Encryption**

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

### 353 **4.1 Attachment Properties**

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

### 358 **4.2 Agreements**

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

#### 361 **4.2.1 CERT-VALUE**

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

366 The Responder MUST have access to the Private key corresponding to the Public key in the  
367 certificate.

#### 368 **4.2.2 Signature Trust Root**

369 There is no digital signature operation for this scenario

### 370 **4.3 Parameters**

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

373 No parameters are required.

### 374 **4.4 General Message Flow**

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

376 This contract covers a request/response MEP over the HTTP binding. SOAP 1.1 MUST be used.  
377 The SOAP envelope MUST be wrapped in a SOAP Message Package as specified by **[SwA]**.

378 The Content-Transfer-Encoding for the encrypted attachment MUST be base64. As required by  
379 SOAP 1.1, the SOAPAction HTTP header MUST be present. Any value, including a null string  
380 may be used. The recipient SHOULD ignore the value. The request contains an encrypted SOAP  
381 attachment. The attachment is encrypted with a random symmetric key, which is encrypted using  
382 a public key certificate. The certificate used for the encryption is provided to the Requestor out of  
383 band. The Responder decrypts the attachment using the symmetric key which is decrypted with  
384 the matching private key. If no errors are detected it returns the response without any security  
385 properties.

## 386 4.5 First Message - Request

### 387 4.5.1 Message Elements and Attributes

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

392

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

### 393 4.5.2 Message Creation

#### 394 4.5.2.1 Security

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

#### 396 4.5.2.2 BinarySecurityToken

397 The ValueType MUST be X509v3. The EncodingType MUST be Base64Binary. The token MUST  
398 be labeled with an Id so it can be referenced by the signature. The value MUST be a Public Key

399 certificate suitable for symmetric key encryption. The certificate SHOULD NOT have a KeyUsage  
400 extension. If it does contain a KeyUsage extension, it SHOULD include the values of  
401 keyEncipherment and dataEncipherment. The Responder must have access to the private key  
402 corresponding to the public key in the certificate.

### 403 **4.5.2.3 EncryptedKey**

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

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

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

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

### 412 **4.5.2.4 EncryptedData**

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

### 417 **4.5.2.5 EncryptionMethod**

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

### 419 **4.5.2.6 CipherData**

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

424

### 425 **4.5.2.7 Body**

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

### 427 **4.5.2.8 Ping**

428 The Ping element should contain the scenario number and the name of the entity performing the  
429 request.

### 430 **4.5.2.9 Post Operation Attachment Properties**

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

### 436 4.5.3 Responder Message Processing

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

#### 440 4.5.3.1 Security

#### 441 4.5.3.2 BinarySecurityToken

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

#### 444 4.5.3.3 EncryptedKey

445 The random key contained in the CipherData MUST be decrypted using the private key  
446 corresponding to the certificate specified by the SecurityTokenReference, using the specified  
447 algorithm.

#### 448 4.5.3.4 EncryptedData

449 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted  
450 symmetric key.

#### 451 4.5.3.5 Attachment

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

### 453 4.5.4 Example (Non-normative)

454 Here is an example request.

```
455 Content-Type: multipart/related; boundary="enc-example"; type="text/xml"
456 --enc-example
457 Content-Type: text/xml; charset=utf-8
458
459 <?xml version="1.0" encoding="utf-8" ?>
460 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
461 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
462 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
463 <soap:Header>
464 <wsse:Security soap:mustUnderstand="1"
465 xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
466 secext-1.0.xsd">
467
468 <!-- This certificate is used for symmetric key encryption -->
469 <wsse:BinarySecurityToken
470 Value="MIIE...hk"
471 EncodingType="wsse:Base64Binary"
472 xmlns:wsu="http://docs.oasis
473 open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
474 wsu:Id="myEncCert">MIIE...hk</wsse:BinarySecurityToken>
475
476 <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
477 <xenc:EncryptionMethod
478 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
479 <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
480 <wsse:SecurityTokenReference>
481 <wsse:Reference URI="#myEncCert" />
482 </wsse:SecurityTokenReference>
483 </KeyInfo>
484 <xenc:CipherData>
485 <xenc:CipherValue>dNYS...fQ</xenc:CipherValue>
486 </xenc:CipherData>
487 <xenc:ReferenceList>
```

```

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

```

526

## 527 4.6 Second Message - Response

### 528 4.6.1 Message Elements and Attributes

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

532

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

533

### 534 4.6.2 Message Creation

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



### 537 **4.6.2.1 Security**

538 There are no security properties on the response message

### 539 **4.6.2.2 Body**

540 The body element MUST be not be signed or encrypted

### 541 **4.6.3 Message Processing**

542 The response is passed to the application without modification.

### 543 **4.6.4 Example (Non-normative)**

544 Here is an example response.

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

### 555 **4.7 Other processing**

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

#### 557 **4.7.1 Requester**

558 No additional processing is required.

#### 559 **4.7.2 Responder**

560 No additional processing is required.

### 561 **4.8 Expected Security Properties**

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

---

## 565 **5 Scenario #3 – Attachment Signature and** 566 **Encryption**

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

### 572 **5.1 Attachment Properties**

573 This section specifies the attachment properties BEFORE security operations are applied. The  
574 Content-Type of the attachment MUST be text/xml. The Content-Transfer-Encoding MUST be  
575 8bit ASCII (8bit). The attachment MUST have a Content-Id header that uniquely identifies the  
576 attachment. The generation of the Content-Id header is out of scope. An example of what the  
577 attachment may look like before encryption and signing is shown as follows. This example is non-  
578 normative.

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

586

### 587 **5.2 Agreements**

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

#### 590 **5.2.1 CERT-VALUE**

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

595 The Responder MUST have access to the private key corresponding to the public key in the  
596 certificate.

#### 597 **5.2.2 Signature Trust Root**

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

### 601 **5.3 Parameters**

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

604 No parameters are required.

## 605 5.4 General Message Flow

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

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

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

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

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

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

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

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

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

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

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

617 Response with no security properties.

## 618 5.5 First Message - Request

### 619 5.5.1 Message Elements and Attributes

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

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

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

623 appear in the order specified, except as noted.

624

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

## 625 **5.5.2 Message Creation**

### 626 **5.5.2.1 Security**

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

### 628 **5.5.2.2 BinarySecurityToken**

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

### 634 **5.5.2.3 EncryptedKey**

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

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

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

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

### 644 **5.5.2.4 EncryptedData**

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

### 649 **5.5.2.5 EncryptionMethod**

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

### 651 **5.5.2.6 CipherData**

652 The CipherData MUST refer to the encrypted attachment with a CipherReference element. The  
653 CipherReference element MUST refer to the attachment using a URI with a cid scheme. The  
654 CipherReference must have a Transforms child with a single Transform sub child with the value  
655 of #Attachment-Content-Only-Transform.

### 656 **5.5.2.7 BinarySecurityToken**

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

### 663 **5.5.2.8 Signature**

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

#### 666 **5.5.2.8.1 SignedInfo**

667 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST  
668 be RSA-SHA1.

#### 669 **5.5.2.8.2 Reference**

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

#### 673 **5.5.2.8.3 SignatureValue**

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

#### 676 **5.5.2.8.4 KeyInfo**

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

### 680 **5.5.2.9 Body**

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

### 682 **5.5.2.10 Post Operation Attachment Properties**

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

### 688 **5.5.3 Responder Message Processing**

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

#### 692 **5.5.3.1 Security**

##### 693 **5.5.3.2 BinarySecurityToken**

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

##### 696 **5.5.3.3 EncryptedKey**

697 The random key contained in the CipherData MUST be decrypted using the private key  
698 corresponding to the certificate specified by the SecurityTokenReference, using the specified  
699 algorithm.

##### 700 **5.5.3.4 EncryptedData**

701 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted  
702 symmetric key.

##### 703 **5.5.3.5 Attachment**

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

##### 705 **5.5.3.6 BinarySecurityToken**

706 The certificate in the token MUST be validated. The Subject of the certificate MUST be an  
707 authorized entity. The public key in the certificate MUST be retained for verification of the  
708 signature.

##### 709 **5.5.3.7 Signature**

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

##### 712 **5.5.3.8 Attachment**

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

### 714 **5.5.4 Example (Non-normative)**

715 Here is an example request.

```
716 Content-Type: multipart/related; boundary="enc-sig-example"; type="text/xml"
717 --enc-sig-example
718 Content-Type: text/xml; charset=utf-8
719
720 <?xml version="1.0" encoding="utf-8" ?>
721 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
722 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
723 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
724 <soap:Header>
725 <wsse:Security soap:mustUnderstand="1"
726 xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
727 secext-1.0.xsd">
728
729 <!-- This certificate is used for symmetric key encryption -->
```

```

730 <wsse:BinarySecurityToken
731   ValueType="wsse:X509v3"
732   EncodingType="wsse:Base64Binary"
733   xmlns:wsu="http://docs.oasis
734     open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
735   wsu:Id="myEncCert">MII...hk</wsse:BinarySecurityToken>
736
737 <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
738   <xenc:EncryptionMethod
739     Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
740   <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
741     <wsse:SecurityTokenReference>
742       <wsse:Reference URI="#myEncCert" />
743     </wsse:SecurityTokenReference>
744   </KeyInfo>
745   <xenc:CipherData>
746     <xenc:CipherValue>dNYS...fQ=</xenc:CipherValue>
747   </xenc:CipherData>
748   <xenc:ReferenceList>
749     <xenc:DataReference URI="#encrypted-signed-attachment" />
750   </xenc:ReferenceList>
751 </xenc:EncryptedKey>
752
753 <!-- The EncryptedData portion here refers to content of the attachment -->
754
755   <xenc:EncryptedData wsu:Id="encrypted-signed-attachment"
756     Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
757     1.0#Attachment-Content-Only" MimeType="text/xml">
758     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
759     <xenc:EncryptionMethod
760       Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc" />
761     <xenc:CipherData>
762       <xenc:CipherReference URI="cid:encsignexample">
763         <xenc:Transforms>
764           <ds:Transform
765             Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
766             profile-1.0#Attachment-Content-Only-Transform" />
767           <ds:Transform
768             Algorithm="http://www.w3.org/2000/09/xmldsig#base64" />
769         </xenc:Transforms>
770       </xenc:CipherReference>
771     </xenc:CipherData>
772   </xenc:EncryptedData>
773
774 <!-- This certificate is used to verify the signature -->
775 <wsse:BinarySecurityToken ValueType="wsse:X509v3"
776   EncodingType="wsse:Base64Binary" xmlns:wsu="http://docs.oasis
777   open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
778   wsu:Id="mySigCert">MII...hk</wsse:BinarySecurityToken>
779
780 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
781   <SignedInfo>
782     <CanonicalizationMethod
783       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
784     <SignatureMethod
785       Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
786     <Reference URI="cid:encsignexample">
787       <Transforms>
788         <Transform Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-
789         2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform"/>
790       </Transforms>
791       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
792       <DigestValue>QTV...dw=</DigestValue>
793     </Reference>
794   </SignedInfo>
795   <SignatureValue>H+x0...gUw=</SignatureValue>
796   <KeyInfo>
797     <wsse:SecurityTokenReference>
798       <wsse:Reference URI="#mySigCert" />
799     </wsse:SecurityTokenReference>
800   </KeyInfo>

```

```

801     </Signature>
802     </wsse:Security>
803 </soap:Header>
804 <soap:Body>
805   <Ping xmlns="http://xmlsoap.org/Ping">
806     <text>Acme Corp. - Scenario #3</text>
807   </Ping>
808 </soap:Body>
809 </soap:Envelope>
810 --enc-sig-example
811 Content-Type: application/octet-stream
812 Content-Id: <encsignexample>
813 Content-Transfer-Encoding: base64
814
815 FEWMMIIfc93ASjfdjsa358sa98xsjcx

```

816

## 817 5.6 Second Message - Response

### 818 5.6.1 Message Elements and Attributes

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

822

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

823

### 824 5.6.2 Message Creation

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

#### 827 5.6.2.1 Security

828 There are no security properties on the response message

#### 829 5.6.2.2 Body

830 The body element MUST be not be signed or encrypted

### 831 5.6.3 Message Processing

832 The response is passed to the application without modification.

### 833 5.6.4 Example (Non-normative)

834 Here is an example response.

```

835 <?xml version="1.0" encoding="utf-8" ?>
836 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
837   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
838   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
839   <soap:Body>

```



```
840 <PingResponse xmlns="http://xmlsoap.org/Ping">
841 <text> Acme Corp. - Scenario #3</text>
842 </PingResponse>
843 </soap:Body>
844 </soap:Envelope>
```

845

## 846 **5.7 Other processing**

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

### 848 **5.7.1 Requester**

849 No additional processing is required.

### 850 **5.7.2 Responder**

851 No additional processing is required.

## 852 **5.8 Expected Security Properties**

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

---

856 **6 Scenario #4 – Attachment Signature and**  
857 **Encryption with MIME Headers**

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

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

867 **6.1 Attachment Properties**

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

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

880 **6.2 Agreements**

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

883 **6.2.1 CERT-VALUE**

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

888 The Responder MUST have access to the private key corresponding to the public key in the  
889 certificate.

890 **6.2.2 Signature Trust Root**

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

894 **6.3 Parameters**

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

897 No parameters are required.

## 898 6.4 General Message Flow

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

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

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

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

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

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

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

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

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

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

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

910 Response with no security properties.

## 911 6.5 First Message - Request

### 912 6.5.1 Message Elements and Attributes

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

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

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

916 appear in the order specified, except as noted.

917

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

## 918 **6.5.2 Message Creation**

### 919 **6.5.2.1 Security**

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

### 921 **6.5.2.2 BinarySecurityToken**

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

### 927 **6.5.2.3 EncryptedKey**

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

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

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

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

### 937 **6.5.2.4 EncryptedData**

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

### 942 **6.5.2.5 EncryptionMethod**

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

### 944 **6.5.2.6 CipherData**

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

### 949 **6.5.2.7 BinarySecurityToken**

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

### 956 **6.5.2.8 Signature**

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

#### 959 **6.5.2.8.1 SignedInfo**

960 The CanonicalizationMethod MUST be Exclusive Canonicalization. The SignatureMethod MUST  
961 be RSA-SHA1.

#### 962 **6.5.2.8.2 Reference**

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

#### 966 **6.5.2.8.3 SignatureValue**

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

#### 969 **6.5.2.8.4 KeyInfo**

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

### 973 **6.5.2.9 Body**

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

### 975 **6.5.2.10 Post Operation Attachment Properties**

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

### 981 **6.5.3 Responder Message Processing**

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

#### 985 **6.5.3.1 Security**

#### 986 **6.5.3.2 BinarySecurityToken**

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

#### 989 **6.5.3.3 EncryptedKey**

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

#### 993 **6.5.3.4 EncryptedData**

994 The attachment referred to by the EncryptedData MUST be decrypted using the encrypted  
995 symmetric key.

#### 996 **6.5.3.5 Attachment**

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

#### 998 **6.5.3.6 BinarySecurityToken**

999 The certificate in the token MUST be validated. The Subject of the certificate MUST be an  
1000 authorized entity. The public key in the certificate MUST be retained for verification of the  
1001 signature.

#### 1002 **6.5.3.7 Signature**

1003 The attachment MUST be verified against the signature using the specified algorithms and  
1004 transforms and the retained public key.

#### 1005 **6.5.3.8 Attachment**

1006 After the attachment's signature has been verified, it should be passed to the application

### 1007 **6.5.4 Example (Non-normative)**

1008 Here is an example request.

```
1009 Content-Type: multipart/related; boundary="enc-sig-headers-example";  
1010 type="text/xml"  
1011 --enc-sig-headers-example  
1012 Content-Type: text/xml; charset=utf-8  
1013  
1014 <?xml version="1.0" encoding="utf-8" ?>  
1015 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
1016 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1017 xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
1018 <soap:Header>  
1019 <wsse:Security soap:mustUnderstand="1"  
1020 xmlns:wsse="docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-  
1021 secext-1.0.xsd">
```

1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093

```
<!-- 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-headers"
    Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-
1.0#Attachment-Complete" 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:encsign-headers-example">
      <xenc:Transforms>
        <ds:Transform
          Algorithm="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
profile-1.0#Attachment-Content-Only-Transform" />
        </ds: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: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>
```

1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108

```
</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
```

## 1109 6.6 Second Message - Response

### 1110 6.6.1 Message Elements and Attributes

1111 Items not listed in the following table MUST NOT be created or processed. Items marked  
1112 mandatory MUST be generated and processed. Items marked optional MAY be generated and  
1113 MUST be processed if present. Items MUST appear in the order specified, except as noted.  
1114

Name	Mandatory?
Envelope	Mandatory
Body	Mandatory
PingResponse	Mandatory

1115

### 1116 6.6.2 Message Creation

1117 The response message MUST NOT contain a <wsse:Security> header. Any other header  
1118 elements MUST NOT be labeled with a mustUnderstand="1" attribute.

#### 1119 6.6.2.1 Security

1120 There are no security properties on the response message

#### 1121 6.6.2.2 Body

1122 The body element MUST be not be signed or encrypted

### 1123 6.6.3 Message Processing

1124 The response is passed to the application without modification.

### 1125 6.6.4 Example (Non-normative)

1126 Here is an example response.

```
1127 <?xml version="1.0" encoding="utf-8" ?>
1128 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
1129 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1130 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
1131 <soap:Body>
1132 <PingResponse xmlns="http://xmlsoap.org/Ping">
```



```
1133 <text> Acme Corp. - Scenario #4</text>
1134 </PingResponse>
1135 </soap:Body>
1136 </soap:Envelope>
```

1137

## 1138 **6.7 Other processing**

1139 This section describes processing that occurs outside of generating or processing a message.

### 1140 **6.7.1 Requester**

1141 No additional processing is required.

### 1142 **6.7.2 Responder**

1143 No additional processing is required.

## 1144 **6.8 Expected Security Properties**

1145 Use of the service is restricted to authorized parties that sign the attachment. The request  
1146 attachment is protected against modification and interception. The response is not protected in  
1147 any way.

1148

---

1149 **7 References**

1150 **7.1 Normative**

- 1151 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,  
1152 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 1153 **[SwA]** W3C Note, "SOAP Messages with Attachments", 11 December 2000,  
1154 <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-2001211>.
- 1155 **[WSS-SwA]** Hirsch, Frederick, *Web Services Security SOAP Message with Attachments*  
1156 Profile 1.0, OASIS Draft 8 2004

## Appendix A. Ping Application WSDL File

```

1158 <definitions xmlns:tns="http://xmlsoap.org/Ping" xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1159 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"
1160 targetNamespace="http://xmlsoap.org/Ping" name="Ping">
1161 <types>
1162 <schema targetNamespace="http://xmlsoap.org/Ping" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
1163 <import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd" schemaLocation="http://docs.oasis-
1164 open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"/>
1165
1166 <element name="text" type="xsd:string" nillable="true"/>
1167 <complexType name="ping">
1168 <sequence>
1169 <element ref="tns:text"/>
1170
1171 </sequence>
1172 </complexType>
1173 <complexType name="pingResponse">
1174 <sequence>
1175 <element ref="tns:text"/>
1176 </sequence>
1177 </complexType>
1178 <element name="Ping" type="tns:ping"/>
1179 <element name="PingResponse" type="tns:pingResponse"/>
1180 </schema>
1181 </types>
1182 <message name="PingRequest">
1183 <part name="ping" element="tns:ping"/>
1184 </message>
1185 <message name="PingResponse">
1186 <part name="pingResponse" element="tns:PingResponse"/>
1187 </message>
1188 <portType name="PingPort">
1189 <operation name="Ping">
1190 <input message="tns:PingRequest"/>
1191 <output message="tns:PingResponse"/>
1192 </operation>
1193 </portType>
1194 <binding name="PingBinding" type="tns:PingPort">
1195 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1196 <operation name="Ping">
1197 <soap:operation/>
1198 <input>
1199 <soap:body use="literal"/>
1200 </input>
1201 <output>
1202 <soap:body use="literal"/>
1203 </output>
1204 </operation>
1205 </binding>
1206 <service name="PingService">
1207 <port name="Ping1" binding="tns:PingBinding">
1208 <soap:address location="http://localhost:9080/pingservice/Ping1"/>
1209 </port>
1210 <port name="Ping2" binding="tns:PingBinding">
1211 <soap:address location="http://localhost:9080/pingservice/Ping2"/>
1212 </port>
1213 <port name="Ping3" binding="tns:PingBinding">
1214 <soap:address location="http://localhost:9080/pingservice/Ping3"/>
1215 </port>
1216 <port name="Ping4" binding="tns:PingBinding">
1217 <soap:address location="http://localhost:9080/pingservice/Ping4"/>
1218 </port>
1219 <port name="Ping5" binding="tns:PingBinding">
1220 <soap:address location="http://localhost:9080/pingservice/Ping5"/>
1221 </port>
1222 <port name="Ping6" binding="tns:PingBinding">
1223 <soap:address location="http://localhost:9080/pingservice/Ping6"/>
1224 </port>
1225 <port name="Ping7" binding="tns:PingBinding">
1226 <soap:address location="http://localhost:9080/pingservice/Ping7"/>
1227 </port>
1228 </service>
1229 </definitions>
1230
1231

```

1232

---

## Appendix B. - Revision History

1233

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.

1234

---

1235

## Appendix C. Notices

1236 OASIS takes no position regarding the validity or scope of any intellectual property or other rights  
1237 that might be claimed to pertain to the implementation or use of the technology described in this  
1238 document or the extent to which any license under such rights might or might not be available;  
1239 neither does it represent that it has made any effort to identify any such rights. Information on  
1240 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS  
1241 website. Copies of claims of rights made available for publication and any assurances of licenses  
1242 to be made available, or the result of an attempt made to obtain a general license or permission  
1243 for the use of such proprietary rights by implementors or users of this specification, can be  
1244 obtained from the OASIS Executive Director.

1245 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
1246 applications, or other proprietary rights which may cover technology that may be required to  
1247 implement this specification. Please address the information to the OASIS Executive Director.

1248 **Copyright © OASIS Open 2004. All Rights Reserved.**

1249 This document and translations of it may be copied and furnished to others, and derivative works  
1250 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
1251 published and distributed, in whole or in part, without restriction of any kind, provided that the  
1252 above copyright notice and this paragraph are included on all such copies and derivative works.  
1253 However, this document itself does not be modified in any way, such as by removing the  
1254 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS  
1255 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual  
1256 Property Rights document must be followed, or as required to translate it into languages other  
1257 than English.

1258 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
1259 successors or assigns.

1260 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
1261 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO  
1262 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE  
1263 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
1264 PARTICULAR PURPOSE.