

# nehta

---

## **Web Services Profile**

Version 3.1 — 30 June 2009

---

**National E-Health Transition Authority Ltd**

Level 25

56 Pitt Street

Sydney, NSW, 2000

Australia.

[www.nehta.gov.au](http://www.nehta.gov.au)

**Disclaimer**

NEHTA makes the information and other material (“Information”) in this document available in good faith but without any representation or warranty as to its accuracy or completeness. NEHTA cannot accept any responsibility for the consequences of any use of the Information. As the Information is of a general nature only, it is up to any person using or relying on the Information to ensure that it is accurate, complete and suitable for the circumstances of its use.

**Document Control**

This document is maintained in electronic form. The current revision of this document is located on the NEHTA Web site and is uncontrolled in printed form. It is the responsibility of the user to verify that this copy is of the latest revision.

**Copyright © 2009, NEHTA.**

This document contains information which is protected by copyright. All Rights Reserved. No part of this work may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without the permission of NEHTA. All copies of this document must include the copyright and other information contained on this page.

# Table of contents

<b>Table of contents</b> .....	<b>iii</b>
<b>Document information</b> .....	<b>v</b>
Change history .....	v
<b>1 Introduction</b> .....	<b>1</b>
1.1 Background .....	1
1.1.1 Document history .....	1
1.2 Purpose .....	1
1.3 Scope .....	1
1.3.1 Content .....	1
1.3.2 Purpose .....	1
1.3.3 Related publications .....	2
1.3.4 Document map .....	2
1.3.5 Audience .....	3
1.3.6 Usage .....	3
1.3.7 Artefacts .....	4
1.4 Status .....	4
1.5 Normative references .....	5
1.6 Definitions, acronyms, abbreviations .....	6
1.7 Conformance .....	7
1.8 Overview .....	7
<b>2 Profiles</b> .....	<b>8</b>
2.1 Web services base profile .....	8
2.2 WS-Security profile .....	8
2.3 TLS security profile .....	8
<b>3 Web services specifications</b> .....	<b>9</b>
3.1 Web service description .....	9
3.1.1 WSDL 1.1 .....	9
3.1.2 Separation of interface and bindings .....	9
3.1.3 Namespace values .....	10
3.2 Web service policies .....	12
3.2.1 WS-Policy 1.5 – Framework .....	12
3.2.2 WS-SecurityPolicy 1.2 .....	13
3.2.3 WS-Addressing 1.0 – Metadata .....	13
<b>4 Transport</b> .....	<b>14</b>
4.1 HyperText Transport Protocol .....	14
4.1.1 HTTP 1.1 .....	14
4.1.2 HTTP persistent connections .....	14
<b>5 Protocol</b> .....	<b>15</b>
5.1 SOAP .....	15
5.1.1 SOAP 1.2 .....	15
5.1.2 Document literal encoding .....	15
5.1.3 SOAP Action .....	16
5.1.4 Wrapped convention .....	17
5.1.5 Request-response message exchange pattern .....	18
5.1.6 Fault and error behaviour .....	19
<b>6 WS-Security security</b> .....	<b>21</b>
6.1 PKI .....	21
6.1.1 Use of PKI .....	21
6.1.2 Subject Key Identifier of X.509v3 certificates .....	22
6.1.3 Key usage .....	23

6.1.4	Certificates with identifiers .....	24
6.2	WS-Security .....	24
6.2.1	WS-Security 1.1 .....	24
6.2.2	WS-Security timestamp .....	25
6.2.3	Digital signatures .....	30
6.2.4	Encryption .....	32
6.2.5	Sign before encryption.....	34
6.2.6	Algorithms .....	36
6.2.7	Transmission of certificates .....	38
<b>7</b>	<b>Metadata .....</b>	<b>43</b>
7.1	WS-Addressing.....	43
7.1.1	Use of WS-Addressing 1.0 .....	43
7.1.2	WS-Addressing Action.....	44
7.1.3	WS-Addressing MessageID .....	45
7.1.4	WS-Addressing in SOAP requests .....	46
7.1.5	WS-Addressing in SOAP responses .....	48
7.1.6	WS-Addressing in SOAP faults.....	48
<b>8</b>	<b>TLS security.....</b>	<b>50</b>
8.1	PKI for TLS .....	50
8.1.1	RSA certificates .....	50
8.1.2	Key usage.....	50
8.1.3	Certificates with identifiers .....	51
8.2	Transport Layer Security.....	51
8.2.1	Protocol.....	51
8.2.2	Mutual authentication .....	52
8.2.3	Cipher suites .....	52
	<b>Appendix A: Informative references .....</b>	<b>54</b>
	<b>Appendix B: NEHTA SOAP faults .....</b>	<b>55</b>
B.1	SOAP faults .....	55
B.2	Standard error codes .....	56
B.2.1	Service errors.....	56
B.2.2	Certificate type errors.....	56
B.2.3	Security errors .....	56
B.2.4	Bad request errors .....	57
B.3	XML Schema.....	58
	<b>Appendix C: Change log.....</b>	<b>59</b>

# Document information

## Change history

Version	Date	Comments
1.0	2006-01-31	Draft for comment
2.0	2006-11-20	Final
3.0	2008-12-01	Final
3.1	2009-06-30	Final

This page is intentionally blank.

# 1 Introduction

## 1.1 Background

The National E-Health Transition Authority (NEHTA) has recommended Web services as the mechanism for communication between organisations in Australia's e-health environment.

The Web services standards by themselves are not sufficient to ensure different systems can be integrated together to exchange information. Those standards are very flexible, which allows them to be implemented in potentially incompatible ways. This profile was created to address those limitations.

### 1.1.1 Document history

This document was previously called the *Web Services Standards Profile v2.0* [WSSP2006].

Version 3.0 of this document incorporates material from the *Guidelines for Implementing Interoperable Web Services* [GIIWS2007]. That guidelines document has now been deprecated.

## 1.2 Purpose

This document describes the NEHTA profiles of the Web services standards.

A profile specifies a set of obligations that must be followed. These obligations identify standards to use and how they are to be applied.

Profiles provide a clear definition of what must be done. Profiles are necessary because often there are alternative standards that can be used and those standards can be interpreted and used in different ways. Profiles have been defined to enable different implementations to be integrated together to exchange information. These profiles have been developed by taking into account the standards and practical implementations of those standards.

The aim of the profiles is to define a common standard that implementations can conform to for the purposes of achieving products that are interoperable and can be integrated together. The Web Services Profile is one part of a larger process that is needed to achieve interoperability and integration.

## 1.3 Scope

### 1.3.1 Content

This document focuses on applying the Web services standards. It covers the specification of Web services interfaces using WSDL, and its realisation with SOAP messages. It also defines profiles for securing Web services.

This document does not cover higher level service design. It also does not cover lower level network details. It does not cover how to implement the profile using any particular programming language or toolkit.

### 1.3.2 Purpose

Web services is suitable for communicating structured data for invoking data services. It is not intended to be used for other forms of communication, such as transmitting large data sets and streaming multimedia, for which other technologies are better suited.

Web services is for use for external organisation to organisation communications in the e-health environment. It can, but does not have to, be used for internal communications within an organisation.

### 1.3.3 Related publications

A familiarity with the NEHTA *Concepts and Patterns for Implementing Services* is useful for understanding the motivation behind some of the criteria in this document [CPIS2008].

This document complements the profiles that other organisations have published to improve the interoperability of Web services. The best known of these profiles are from the Web Services Interoperability (WS-I) group. The group has published the *Basic Profile 1.1* [BP2006] and *Basic Security Profile 1.0* [BSP2007]. Where applicable, NEHTA tries to adopt the recommendations in these profiles. However, in general, the recommendations in the WS-I profiles are low-level. The topics that they cover mostly concern Web services toolkit developers, whereas the NEHTA profiles are aimed at programmers using Web services toolkits. Additionally, the NEHTA profile contains standards which have not been profiled by WS-I. For instance, the NEHTA profile is based on the SOAP 1.2 standard. SOAP 1.2 is covered only in WS-I Basic Profile 2.0 [BP2007], which at the time of writing is only at working draft stage.

### 1.3.4 Document map

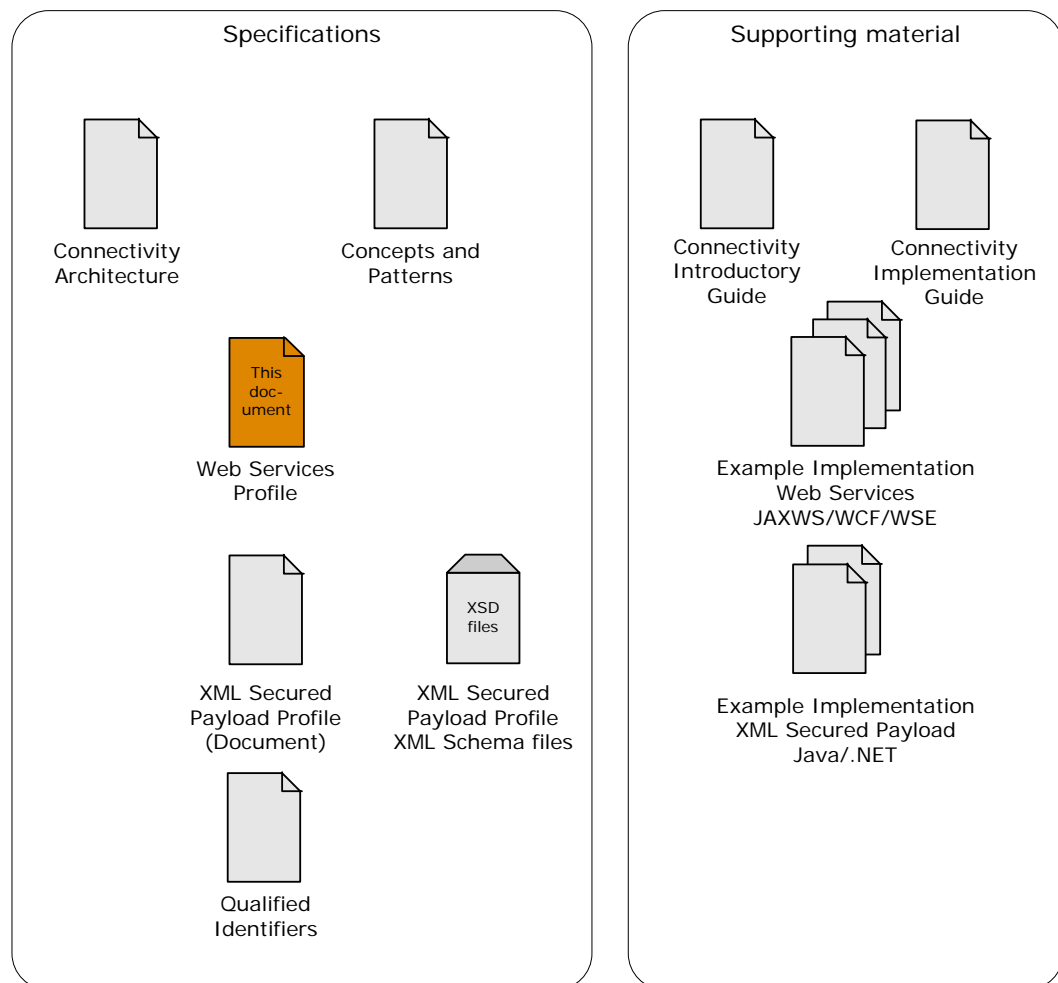


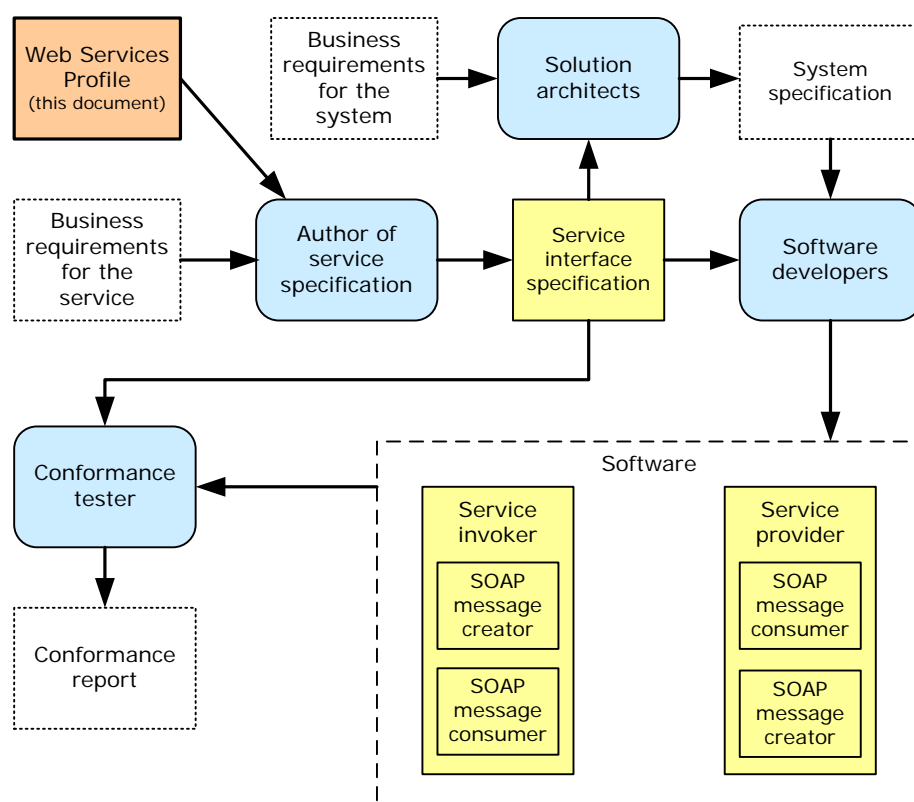
Figure 1: Document map

### 1.3.5 Audience

This document is intended for:

- **Authors of service interface specifications**, who incorporate the profiles into the service interface specifications they produce; and
- **Solution architects**, who incorporate the service interface specifications into the system specifications they design;
- **Software developers**, who create software programs which conform to the system specification and service interface specification; and
- **Conformance testers**, who test software to determine how it conforms to the service interface specifications and the profiles that it uses.

The different audiences for this document are illustrated in Figure 2. Although the solution architects, software developers and conformance testers do not directly use the profiles, they need to use it because it will be referenced by the service interface specification.



**Figure 2: Web Services Profile users and artefacts**

The reader is expected to have a detailed technical knowledge of PKI and Web services at the technical implementation and protocol level.

### 1.3.6 Usage

The main use of this document is for developing service interface specifications. The service interface specifications describe what the service is. That specification would reference profiles from this document.

Implementations will have to refer to this document to comply with those service interface specifications.

### 1.3.7 Artefacts

The criteria in this document are categorised according to the artefacts which they apply to.

The artefacts used in this document are:

- **Service interface specifications**

These obligations also indirectly apply to both service invokers and service providers, because both programs must conform to the service interface specifications. They also indirectly apply to SOAP message creators and SOAP message consumers, because service invokers and service providers are also SOAP message creators and SOAP message consumers.

These obligations will apply to service invokers and service providers who implement those service interface specifications. It applies to them indirectly, because they following the service interface specifications.

- **Service invokers**

Service invokers use a service. The service invokers create request messages. They also consume response messages and fault messages.

A service invoker is also a SOAP message creator, because they create SOAP request messages. A service invoker is also a SOAP message consumer, because they consume SOAP response messages and SOAP faults.

- **Service providers**

Service providers offer a service. The service providers consume request messages. They also create response messages and fault messages.

A service provider is also a SOAP message creator, because they create SOAP response messages and SOAP faults. A service provider is also a SOAP message consumer, because they consume SOAP request messages.

- **SOAP message creators**

SOAP message creators generate SOAP messages and faults.

SOAP message creators are service invokers generating request messages, and service providers generating response messages and fault messages.

- **SOAP message consumers**

SOAP message consumers process SOAP messages and faults.

SOAP message consumers are service providers processing request messages, and service invokers processing response messages and fault messages.

These artefacts, and how they relate to each other, are illustrated in Figure 2. That figure also shows the audiences for this document, and their relationships to the artefacts—which artefacts they use and which artefacts they produce.

## 1.4 Status

This document is a final release.

## 1.5 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- [CAIS2008] NEHTA, *Concepts and Patterns for Implementing Services*, version 2.0, 1 December 2008.
- [MAME2005] Anne Manes, *The “wrapped” document/literal convention*, 20 March 2005,  
<http://atmanes.blogspot.com/2005/03/wrapped-documentliteral-convention.html>
- [RFC2119] IETF, *RFC 2119: Keywords for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997,  
<http://ietf.org/rfc/rfc2119.txt>
- [RFC2141] IETF, *RFC 2141: URN Syntax*, R. Moats, May 1997,  
<http://ietf.org/rfc/rfc2141.txt>
- [RFC2246] IETF, *RFC 2246: The TLS Protocol Version 1.0*, T. Dierks and C. Allen, January 1999  
<http://ietf.org/rfc/rfc2246.txt>
- [RFC2616] IETF, *RFC 2616: HyperText Transfer Protocol – HTTP/1.1*, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999,  
<http://ietf.org/rfc/rfc2616.txt>
- [RFC4122] IETF, *RFC 4122: A Universally Unique Identifier (UUID) URN Namespace*, P. Leach, M. Mealling, R. Salz, July 2005,  
<http://ietf.org/rfc/rfc4122.txt>
- [RFC4346] IETF, *RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1*, T. Dierks and E. Rescorla, April 2006,  
<http://ietf.org/rfc/rfc4346.txt>
- [RFC5246] IETF, *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2*, T. Dierks and E. Rescorla, August 2008,  
<http://ietf.org/rfc/rfc5246.txt>
- [SMCC2008] NEHTA, *Secure Messaging Common Components*, 2008.
- [SOAP2003b] W3C, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)* W3C Recommendation, 27 April 2007,  
<http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>
- [SOAP2003c] W3C, *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*, W3C Recommendation, 27 April 2007,  
<http://www.w3.org/TR/2007/REC-soap12-part2-20070427/>
- [SSL30] The SSL Protocol, Version 3.0, Alan Freier, Philip Karlton and Paul Kocher, IETF Internet draft, 18 November 1996.
- [WSA2006] W3C, *Web Services Addressing 1.0 – Core*, W3C Recommendation, 9 May 2006,  
<http://www.w3.org/TR/2006/REC-ws-addr-core-20060509>.
- [WSAM2007] W3C, *Web Services Addressing 1.0—Metadata*, W3C Recommendation, 4 September 2007,  
<http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904>
- [WSDL2001] W3C, *Web Services Description Language (WSDL) 1.1*, W3C Note, 15 March 2001,  
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

- [WSPL2007] W3C, *Web Services Policy 1.5 – Framework*, W3C Recommendation, 4 September 2007, <http://www.w3.org/TR/2007/REC-ws-policy-20070904>.
- [WSS2006] OASIS, *Web Services Security: SOAP Message Security 1.1*, OASIS Standard, 1 February 2006, <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [WSSPL2007] OASIS, *WS-SecurityPolicy 1.2*, OASIS Standard, 1 July 2007, <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf>.
- [XCTP2006] OASIS, *Web Services Security X.509 Certificate Token Profile 1.1*, OASIS Standard Specification, 1 February 2006, <http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-x509TokenProfile.pdf>
- [X509-2005] ITU-T, *Information technology—Open Systems Interconnection—The Directory: Public-key and attribute certificate frameworks*, ITU-T Recommendation X.509, August 2005.
- [XML2006] W3C, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, 16 August 2006, <http://www.w3.org/TR/2006/REC-xml-20060816>

Non-normative references can be found in Appendix A: “Informative references”.

## 1.6 Definitions, acronyms, abbreviations

Profile	A profile identifies a set of standards and how they are to be used.
Service invoker	A program that makes calls on the Web service offered by a service provider. It acts as a client. A service invoker functions as a SOAP message creator (because it creates request messages) and as a SOAP message consumer (because it consumes response and fault messages).
Service provider	A program that offers a Web service. It acts as a server. A service provider functions as a SOAP message consumer (because it consumes request messages) and as a SOAP message creator (because it creates responses and faults).
SOAP	A protocol for using XML-based messages to exchange structured data in a distributed computing environment.
SOAP message creator	A program that creates and sends SOAP messages.
SOAP message consumer	A program that receives and processes SOAP messages.
Toolkit	A set of tools that assists the implementation of Web services protocols. These tools usually include a programming library, code generation tools, deployment and configuration programs. A toolkit can be provided with a platform or as a third party component.

DMZ	Demilitarized Zone
HTTP	HyperText Transfer Protocol
MEP	Message Exchange Pattern
NTP	Network Time Protocol
PKI	Public Key Infrastructure
RPC	Remote Procedure Call
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
WSDL	Web Services Definition Language
XML	Extensible Markup Language

## 1.7 Conformance

To conform to the profile, an artefact **MUST** satisfy every criteria in the profile associated with that artefact.

The keywords **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** in this document are to be interpreted as described in IETF's RFC 2119 [RFC2119].

## 1.8 Overview

This document consists of two parts:

- Chapter 2 defines the Web services profiles.
- The remaining chapters define the criteria that are referenced by the profiles in Chapter 2.

The criteria are grouped into sections which relate to the same topic. Within each of these sections, there are subsections which classify the criteria according to which artefact they apply to.

Each topic section can also contain notes and examples. These are non-normative. They provide supporting information to assist in understanding the obligations, but are not an official part of the profile.

Each criterion is numbered so that it can be uniquely referenced. The number is of the form "WS n.n.n.n-m" where n.n.n.n is the section it appears in and m uniquely identifies it in the section.

## 2 Profiles

A profile specifies a set of obligations that must be followed. These obligations identify standards to use and how they are to be applied.

Three profiles are defined in this document:

- Web services base profile;
- WS-Security profile; and
- TLS security profile.

The service interface specification **MUST** specify which profile(s) it uses. For example, a service specification could choose to use the *Web services base profile* plus one of the security profiles.

### 2.1 Web services base profile

This profile consists of the following obligations:

- All criteria from chapter 3 “Web services specifications”;
- All criteria from chapter 4 “Transport”;
- All criteria from chapter 5 “Protocol”; and
- All criteria from chapter 7 “Metadata”.

### 2.2 WS-Security profile

This profile consists of the following obligations:

- All criteria from chapter 6 “WS-Security security”.

### 2.3 TLS security profile

This profile consists of the following obligations:

- All criteria from chapter 8 “TLS security”.

# 3 Web services specifications

## 3.1 Web service description

### 3.1.1 WSDL 1.1

#### 3.1.1.1 Conformance criteria on service interface specifications

WS 3.1.1.1-1 Service interface specifications **MUST** provide a description of the technical service using the Web Services Description Language (WSDL) 1.1.

WSDL 1.1 is defined by [WSDL2001].

#### 3.1.1.2 Notes (non-normative)

WSDL is a machine readable form for describing what a service does and how to invoke it. The description can be used as part of the formal documentation of the service, and it can also be used as input for development tools and programs.

WSDL can also be used to indicate where a service instance is located, but this feature will not be used in this profile. This is because WSDL is being used to define the design time features of a service interface. In this profile it is not being used to describe a run time service instance.

WSDL 1.1 is a W3C Note. It is not an official standard, but it is widely used and supported by current tools and products. Note: WSDL 1.0 was an unofficial draft that has been superseded by WSDL 1.1.

Note that a WSDL document is only one part of the service interface specification. The WSDL is a major part of the service interface specifications, but there are aspects of the service interface which the WSDL cannot represent. Those other aspects of the service interface will need to be represented in other machine readable formats or using natural language.

Programmers should be aware that the WSDL is primarily being used as a specification document. There is no obligation to use the WSDL file to generate computer code, even though most Web services toolkits provide a mechanism to do this. Programmers have the option to implement the service interface using another method—as long as the result conforms to what the WSDL specifies.

### 3.1.2 Separation of interface and bindings

#### 3.1.2.1 Conformance criteria on service interface specifications

WS 3.1.2.1-1 Service interface specifications **SHOULD** separate their WSDL descriptions into at least two files—one file containing the interface elements: types, message, portType; and a separate file containing the binding and service elements.

#### 3.1.2.2 Notes (non-normative)

WSDL can be used to describe both the abstract service interface and the concrete aspects of a particular service binding. Textbook examples of WSDL puts everything into a single WSDL file. This makes it easier to show and describe, but makes it less easy to reuse.

To improve reuse, the concrete aspects need to be separated from the abstract aspects. This would allow the concrete aspects to be modified without needing to modify the abstract aspects.

The file containing the binding and service elements can use the WSDL `import` mechanism to refer to the interface WSDL.

WSDL can also be used to contain deployment information about a particular service instance. For example, the URL where the service instance is running. However, this is not useful when WSDL is being used to specify service interfaces that will be implemented by multiple service instances.

WSDL will be used to specify service interfaces and not to describe a particular service instance. The service instance information can be added by the programmer configuring the toolkit instead of in the service interface WSDL.

### 3.1.2.3 Example (non-normative)

#### WSDL containing service interface information:

```
<?xml version="1.0"?>
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://ns.nehta.gov.au/Example/..."
  targetNamespace="http://ns.nehta.gov.au/Example/..."
  name="...">

  <types>
  ...
  </types>

  <message name="...">
  ...
  </message>
  ...

  <portType name="...">
  ...
  </portType>

</definitions>
```

#### WSDL containing service instance binding information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  ...>
  <import namespace="urn:..." location="... .wsdl"/>

  <binding ...>
  ...
  </binding>

  <service name="...">
  ...
  </service>
</definitions>
```

## 3.1.3 Namespace values

### 3.1.3.1 Conformance criteria on service interface specifications

WS 3.1.3.1-1 Service interface specifications **MUST** specify WS-Addressing Action values as described in *WS-Addressing 1.0 Metadata Recommendation* [WSAM2007].

WS 3.1.3.1-2 Define a unique "service namespace" to identify the service. This service namespaces needs to follow these rules:

- The service namespace **SHOULD** be a URL.
- The service namespace **MAY** be a URN.

WS 3.1.3.1-3 There is a separator character which **MUST** be:

- A slash character (Unicode U+002F) if the service namespace is a URL.
- A colon character (Unicode U+003A) if the service namespace is a URN;

In the following, {serviceNS} is the service namespace, {sep} the separator character, {portType} the port type name, {operation} the operation name, and {faultName} is the name of the fault. The remaining text appears literally as a part of the value.

The WS-Addressing Action values **MUST** follow these rules:

WS 3.1.3.1-4 The target namespace for the WSDL description **MUST** be:

{serviceNS}

WS 3.1.3.1-5 The WS-Addressing Action value for *input* messages **MUST** be:

{serviceNS}{sep}{portType}{sep}{operation}Request

WS 3.1.3.1-6 The WS-Addressing Action value for *output* messages **MUST** be:

{serviceNS}{sep}{portType}{sep}{operation}Response

WS 3.1.3.1-7 The WS-Addressing Action value for *fault* messages **MUST** be:

{serviceNS}{sep}{portType}{sep}{operation}{sep}Fault{sep}{faultName}

WS 3.1.3.1-8 The XML namespace for the wrapper element **MUST** be:

{serviceNS}

### 3.1.3.2 Notes (non-normative)

These conventions are based on the default convention described in the WS-Addressing bindings for WSDL described in [WSAM2007].

Toolkits which do not implement these default conventions will need to have their values overridden with the values specified in these criteria.

See section 5.1.4 "Wrapped convention" for the definition of a wrapper element.

The service namespace must be globally unique. Usually this is achieved by following a particular organisation's policies for assigning namespaces. Those policies ensure that the namespaces used by the organisation can be uniquely associated with the organisation and namespaces are uniquely allocated within the organisation.

### 3.1.3.3 Example (non-normative)

This example uses the following URL as the service namespace:

`http://ns.nehta.gov.au/Example/Svc/ExampleService/1.0`

This service namespace is treated as an opaque value. Although it appears to have components separated by colons, they are not significant to the naming scheme described in this section.

Consider an operation called `bar` with a portType of `foo`. This operation can return a `noID` fault.

Target namespace for the WSDL will be:

`http://ns.nehta.gov.au/Example/Svc/ExampleService/1.0`

The WS-Addressing action for the input message must be:

`http://ns.nehta.gov.au/Example/Svc/ExampleService/1.0/foo/barRequest`

The WS-Addressing action for the output message must be:

`http://ns.nehta.gov.au/Example/Svc/ExampleService/1.0/foo/barResponse`

The WS-Addressing action for the fault message must be:

`http://ns.nehta.gov.au/Example/Svc/ExampleService/1.0/foo/bar/Fault/noID`

The XML namespace for the wrapper element must be:

```
http://ns.nehta.gov.au/Example/Svc/ExampleService/1.0
```

## 3.2 Web service policies

### 3.2.1 WS-Policy 1.5 – Framework

#### 3.2.1.1 Conformance criteria on service interface specifications

WS 3.2.1.1-1 Service interface specifications **MUST** describe the technical policies of the Web services in the WSDL using *WS-Policy 1.5*.

*WS-Policy 1.5* is defined by [WSPL2007].

WS 3.2.1.1-2 Service interface specifications **SHOULD** apply the policy statements within the binding element of the WSDL.

#### 3.2.1.2 Notes (non-normative)

WS-Policy 1.5 is a W3C recommendation that defines a framework for specifying the policies of Web services. The framework acts as a container for policy statements, but it does not define any particular policy statements to use. Other standards specify those policy statements.

WS-Policy provides a way to describe some behaviours of a Web service in a WSDL specification. This improves the role of the WSDL as the technical specification of a Web service.

Policies represent the concrete aspects of a Web service. In keeping with criteria in section 3.1.2, they should be applied to the WSDL binding rather than the WSDL port type.

Currently, only these two standard policy languages should be used:

- WS-Security Policy 1.2 [WSSPL2007] (see criteria in section 3.2.2); and
- WS-Addressing 1.0 (Metadata) [WSAM2007] (see criteria in section 3.2.3).

#### 3.2.1.3 Example (non-normative)

This example shows a policy applied at the root level of the WSDL binding element.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#APolicy" />
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="APolicy">
    ...
  </wsp:Policy>
</definitions>
```

### 3.2.2 WS-SecurityPolicy 1.2

#### 3.2.2.1 Conformance criteria on service interface specifications

WS 3.2.2.1-1 Service interface specifications **MUST** use *WS-SecurityPolicy 1.2* to describe the security constraints and requirements of Web services.

*WS-SecurityPolicy 1.2* is defined by [WSSPL2007].

#### 3.2.2.2 Notes (non-normative)

WS-SecurityPolicy is an OASIS standard for specifying policy assertions for WS-Security and other Web services security standards. It is meant to be used within the WS-Policy framework.

The primary role of specifying WS-SecurityPolicy in a WSDL is for documentation purposes. Some Web service toolkits support it, whereby the WS-SecurityPolicy statements in the WSDL will be used to configure the security of Web services and Web service clients. On the other hand, some toolkits will ignore the WS-SecurityPolicy statements in the WSDL.

Developers will need to check how their toolkit handles WS-SecurityPolicy statements. They must not assume that security is turned on and configured correctly simply because the WSDL from which a Web service was derived from contains WS-SecurityPolicy statements. Additionally, some toolkits support a different version of WS-SecurityPolicy or a subset of it and may reject the standard policies in the WSDL. Developers can modify their copy of the standard WSDL in order to make it usable for their toolkit. Local changes to the standard WSDL are permissible as long as the messages created by SOAP message creators conform to the service interface specification.

### 3.2.3 WS-Addressing 1.0 – Metadata

#### 3.2.3.1 Conformance criteria on service interface specifications

WS 3.2.3.1-1 Service interface specification **MUST** use mechanisms defined in *WS-Addressing 1.0 (Metadata)* to describe addressing requirements in Web services.

*WS-Addressing 1.0 (Metadata)* is defined by [WSAM2007].

#### 3.2.3.2 Notes (non-normative)

*WS-Addressing 1.0 (Metadata)* is a W3C Recommendation that specifies how to define WS-Addressing properties, such as Action, in a WSDL and how to specify the use of addressing in a WSDL using the WS-Policy framework.

Like WS-SecurityPolicy, toolkit support for the mechanisms in *WS-Addressing 1.0 (Metadata)* can vary. Developers cannot assume that addressing is included in SOAP messages from the addressing policy statements in a WSDL.

Criteria for WS-Addressing are documented in section 7.1. Those criteria translate into statements from *WS-Addressing 1.0 (Metadata)* which are then used in WSDL descriptions.

For a discussion about why WS-Addressing is used by the profile, see section 7.1.1.2.

# 4 Transport

## 4.1 HyperText Transport Protocol

### 4.1.1 HTTP 1.1

#### 4.1.1.1 Conformance criteria on service interface specifications

WS 4.1.1.1-1 Service interface specifications **MUST** use the HyperText Transport Protocol (HTTP) version 1.1 as the transport mechanism for Web services.

HTTP 1.1 is defined by [RFC2616].

#### 4.1.1.2 Notes (non-normative)

HTTP 1.1 was chosen because it supports the development of Web services that use an interactive paradigm. It is recognised that the majority of messaging currently conducted in health is non-interactive, but in the future interactive applications will become more prevalent. HTTP 1.1 is a single transport mechanism that is able to support both interactive and non-interactive style Web services.

Currently, HTTP 1.1 is the most widely supported transport mechanism for Web services.

### 4.1.2 HTTP persistent connections

#### 4.1.2.1 Conformance criteria on service invokers

WS 4.1.2.1-1 Service invokers **SHOULD** avoid closing the TCP/IP connection if it can improve performance.

#### 4.1.2.2 Conformance criteria on service providers

WS 4.1.2.2-1 Service providers **SHOULD** avoid closing the TCP/IP connection if it can improve performance.

#### 4.1.2.3 Notes (non-normative)

Under HTTP 1.1, the underlying TCP/IP connection is kept alive by default. If the client or server wishes to close the connection, they can send a "Connection: close" HTTP header. See sections 8.1 and 14.10 of [RFC2616] for details.

These criteria suggest that service invokers and service providers should not send "Connection: close" on every request or response, if it can be helped. Establishing a new TCP/IP connection for every operation would be inefficient if there are multiple operations being invoked consecutively.

However, if the service does not expect multiple operations to occur over the same TCP/IP connection, keeping the connection alive will tie up more resources without improving performance.

# 5 Protocol

## 5.1 SOAP

### 5.1.1 SOAP 1.2

#### 5.1.1.1 Conformance criteria on service interface specifications

WS 5.1.1.1-1 Service interface specifications **MUST** use SOAP 1.2 as the Web services protocol.

SOAP 1.2 is defined by [SOAP2003b] and [SOAP2003c].

#### 5.1.1.2 Notes (non-normative)

SOAP is the standard protocol for Web services. The primer for the SOAP 1.2 specification is [SOAP2003a].

SOAP defines a protocol for exchanging messages in a decentralised and distributed environment. It defines how messages are represented using the XML Infoset [INFO2004], which defines a set of XML data concepts. The Infoset is serialised using the XML 1.0 syntax [XML2006]. It also defines a framework for message processing. SOAP is an extensible protocol which can support different types of payloads, behaviours, and interaction patterns.

SOAP 1.2 is a W3C Recommendation. Although widely used, SOAP 1.1 is only a *de facto* standard and is not a W3C Recommendation [SOAP2000].

### 5.1.2 Document literal encoding

#### 5.1.2.1 Conformance criteria on service interface specifications

WS 5.1.2.1-1 Service interface specifications **MUST** use the “document/literal” WSDL style and usage.

#### 5.1.2.2 Notes (non-normative)

The SOAP binding for WSDL 1.1 permits different encoding styles [WSDL2001]. Any given Web service operation must adopt one particular style to use. Some styles are underspecified and are not as well supported by toolkit implementations. Use of those styles can cause integration problems between different toolkits.

In the “document/literal” style, the structure of the SOAP body and the types used are defined by an XML Schema in the types section in the WSDL.

The literal style is recommended because XML Schema is a mechanism that allows a more expressive description of data than the SOAP encoding mechanism. This style also allows XML Schemas that are developed elsewhere to be used in the messages. The literal style is compliant with the widely implemented WS-I Basic Profile 1.1 [BP2006].

Modern toolkits currently provide better support for the “document/literal” style, over the other possible styles. Some older toolkits only support RPC style operations. However, those older toolkits lack features needed to satisfy other criteria as well.

The document style is recommended over the RPC style, because the RPC approach is tightly coupled and low-level—making it less useful for cross-organisation operations in the e-health environment.

A coarse grain, service oriented approach should be used with Web services that build upon the document exchange metaphor. Aim to send fewer messages, and avoid “chatty” interfaces which frequently send messages.

### 5.1.2.3 Example (non-normative)

This example shows the use of the "document/literal" style in a WSDL description of the service interface.

#### WSDL Definition:

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  ...
  <wsdl:binding ...>
    <wsdl:operation name="...">
      <soap:operation style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="...">
        <soap:fault name="..." use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

## 5.1.3 SOAP Action

### 5.1.3.1 Conformance criteria on service interface specifications

WS 5.1.3.1-1 Service interface specifications **MUST NOT** define SOAP Action values for any operation in the service.

### 5.1.3.2 Conformance criteria on SOAP message consumers

WS 5.1.3.2-1 SOAP message consumers **MUST** ignore any SOAP Action value found in a SOAP message.

### 5.1.3.3 Notes (non-normative)

The function of the SOAP Action has been replaced by the WS-Addressing Action header. The WS-Addressing Action is the preferred mechanism for indicating the Web services operation being invoked. Unlike the WS-Addressing Action, the SOAP Action is placed in HTTP headers rather than in the SOAP header. This means that the SOAP Action value cannot be signed using WS-Security (if it is being used), and therefore is open to man-in-the-middle attacks. These criteria are in line with the WS-I *Basic Security Profile* [BSP2007], which recommends omitting the SOAP Action value from the WSDL.

Older toolkits may provide support for SOAP Action, however, they should not be used. If a SOAP Action value is not available, some toolkits will send an empty string as the value of the SOAP Action. SOAP message consumers need to process these messages, but ignore the SOAP Action values in them, to be compatible with those toolkits.

## 5.1.4 Wrapped convention

### 5.1.4.1 Conformance criteria on service interface specifications

WS 5.1.4.1-1 Service interface specifications **MUST** follow the wrapped convention for structuring the body of the SOAP message.

### 5.1.4.2 Notes (non-normative)

One of the key benefits of the wrapped convention is that it allows the document/literal style of Web services to be used while retaining the ease of use associated with the RPC/literal style implementations.

The wrapped convention has strong support in current toolkits. Its use helps improve compatibility with those toolkits.

The body of SOAP messages can contain arbitrary XML. However, some toolkits expect the body to follow certain conventions. For example, the SOAP body must contain only one XML element, and that element name must uniquely identify the operation. If these conventions are not followed, the toolkits cannot route or process the messages.

The wrapped convention is a constraint on how the SOAP body is structured [BUTE2005]. In the wrapped convention, the WSDL definitions have the following characteristics:

- Input and output messages have only one part.
- Each part references an XML Schema element (i.e. not an XML Schema type). This element is called a wrapper element.
- The wrapper element's type is an XML Schema complex type.
- The wrapper element's type has no attributes.
- The local name of the wrapper element in the input message matches the message's operation name.
- The local name of the wrapper element in the output message matches the message's operation name with "Response" appended to it.

The wrapper element is the root element within the SOAP body. See section 3.1.3 for the conventions on the value of the namespace for the wrapper element.

In the above description, a "part" refers to a WSDL part for the SOAP body. WSDL parts are also used to define SOAP headers, but the wrapped convention does not place any constraints on how the SOAP headers are used.

Some toolkits rely on the XML element in the SOAP body to identify the service operation and routing of messages. There are other mechanisms that can (and should) be used for this purpose (i.e. the WS-Addressing action, which is covered by section 7.1.1.3). However, some earlier toolkits rely on conventions instead of those other mechanisms. Therefore, using the wrapped convention will enable those toolkits to work, without preventing the newer toolkits from using the other mechanisms.

The wrapped convention ensures that the body element contains at most one child element, even though SOAP allows the possibility of multiple child elements. Knowing there is at most one child element can simplify how the body is processed. Having multiple elements in the body is not compliant with the WS-I Basic Profile 1.1 [BP2006], so using the wrapped convention ensures that this need is also satisfied.

Use of the wrapped convention can also simplify programming with toolkits [MAME2005]. The wrapped convention is akin to defining the RPC style using the document/literal style. Some toolkits recognise this similarity and

generate RPC-style method signatures in stub classes, which is easier for developers to program with.

#### 5.1.4.3 Example (non-normative)

This example shows the use of a wrapper element in the WSDL document. The SOAP body refers to a complex type, whose name is the operation name.

##### WSDL Definition:

```
<wsdl:definitions
  targetNamespace="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0">

  <wsdl:types>
    <xsd:schema ...>
      <xsd:element name="sendDS">
        <xsd:complexType ... </xsd:complexType>
      </xsd:element>
      <xsd:element name="sendDSResponse">
        <xsd:complexType ... </xsd:complexType>
      </xsd:element>
      ...
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="sendDSInputMessage">
    <wsdl:part name="parameters" element="tns:aendDS"/>
  </wsdl:message>
  <wsdl:message name="sendDSOutputMessage">
    <wsdl:part name="parameters" element="tns:aendDSResponse"/>
  </wsdl:message>

  <wsdl:portType name="...">
    <wsdl:operation name="sendDS">
      <wsdl:input message="tns:sendDSInputMessage"/>
      <wsdl:output message="tns:sendDSOutputMessage"/>
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

### 5.1.5 Request-response message exchange pattern

#### 5.1.5.1 Conformance criteria on service interface specifications

WS 5.1.5.1-1 Service interface specifications **MUST** use the SOAP Request-Response message exchange pattern.

#### 5.1.5.2 Notes (non-normative)

The SOAP Request-Response message exchange pattern requires that an input and an output message type be provided for the operation type in the WSDL definition. Even if the request has no parameters, it needs to send a SOAP request message with a body containing just the wrapper element (see section 5.1.4) that contains an empty content model. Service providers must generate a SOAP response for every request they receive.

The SOAP Request-Response message exchange pattern was chosen because it is consistent with the service invocation patterns described in the *Concepts and Patterns for Implementing Services* [CAIS2008].

The SOAP Request-Response message exchange pattern is generally well supported by Web service toolkits. The other patterns (i.e. one-way, solicit-response and notification) are generally less mature and increase the risk of interoperability problems. For example, some toolkits use a different MIME type for the HTTP response with the one-way message exchange pattern, which causes the clients to fail.

### 5.1.5.3 Example (non-normative)

This example shows the use of the SOAP Request-Response Message Exchange Protocol (MEP) for every operation. Every operation has both an input and an output message.

#### WSDL Definition:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:tns="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0"
  targetNamespace="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0"
  name="DischargeSummaryReceiver">
  ...

  <!-- tns:SendDischargeSummaryOutMsg is a dummy message -->

  <portType name="DischargeSummaryReceiver">
    <operation name="ping">
      <input message="tns:pingInMsg" .../>
      <output message="tns:pingOutMsg" .../>
    </operation>

    <operation name="sendDischargeSummary">
      <input message="tns:sendDischargeSummaryInMsg" .../>
      <output message="tns:sendDischargeSummaryOutMsg" .../>
      <fault name="..." message="tns:..." .../>
    </operation>

    <operation name="checkStatus">
      <input message="tns:checkStatusInMsg" .../>
      <output message="tns:checkStatusOutMsg" .../>
      <fault name="..." message="tns:..." .../>
    </operation>
    ...
  </portType>
  ...
</definitions>
```

## 5.1.6 Fault and error behaviour

### 5.1.6.1 Conformance criteria on service invokers

- WS 5.1.6.1-1 Service invokers **MUST** expect to receive any SOAP faults, including types of SOAP faults that it does not know about.
- WS 5.1.6.1-2 Service invokers **MUST** expect to receive some SOAP faults that are not signed and/or encrypted using WS-Security (if WS-Security is being used as the security mechanism).

### 5.1.6.2 Conformance criteria on service providers

If a Web service layer error occurs:

- Service providers **MUST** respond with a SOAP fault.
- Service providers **SHOULD** respond with the appropriate NEHTA SOAP fault (as defined in Appendix B: "NEHTA SOAP faults").

- WS 5.1.6.2-1 If an application level error occurs, service providers **MUST** sign and/or encrypt the SOAP fault using WS-Security (if WS-Security is being used as the security mechanism).

- WS 5.1.6.2-2 Service providers **MUST** place the XML element representing the error in the Detail element of the SOAP fault.

### 5.1.6.3 Notes (non-normative)

#### HOW SERVICE INVOKERS HANDLE SOAP FAULTS

To satisfy these criteria, a service invoker needs to handle any type of SOAP fault, including ones it is not expecting. The extent to which it handles the fault will depend on how much it knows about the fault.

If the fault is an application level fault or a NEHTA SOAP fault (as described in Appendix B: “NEHTA SOAP faults”) then it is expected to handle the fault according to the meaning of the fault.

If the fault is unknown to the service invoker, it can only infer that some error has occurred but cannot determine the type of error. They may also log the error for diagnosis by the software vendor. It is not expected that the user will be able to fix the problem, so there is no need to show the details of the unknown fault to them.

The need to handle any type of SOAP fault is necessary because the complete set of possible SOAP faults returned by the service provider cannot be determined beforehand. This is because some Web service toolkits generate some SOAP faults that the programmer has no control over—they cannot change their type and cannot stop them from being generated.

Under certain conditions, the SOAP faults may not even be properly secured. That is, they are not signed and encrypted in the expected manner. This usually happens if using WS-Security and the error makes signing and encrypting the SOAP fault impossible. For example, if the error is that the service invoker’s certificate could not be used. Therefore, the service invoker must also expect to handle SOAP faults which are not properly secured.

Note that unsecured SOAP faults are a potential security risk. They should be processed with caution and not trusted to the same level as secured SOAP faults. It is recommended that they be treated as an indication that a SOAP transport level fault has occurred, but the data in them is not used. It is also recommended that they be logged to help detect security attacks.

#### HOW SERVICE PROVIDERS GENERATE SOAP FAULTS

These criteria are designed as a balance between fully specifying specific SOAP faults and leaving them fully open. Fully specified SOAP faults will allow a much richer and interoperable communications of errors. However, they place greater demands on the implementations.

Most Web service toolkits return proprietary SOAP faults for some errors. Additionally, they give the programmer little or no control of these built-in SOAP faults: the programmer cannot change them and they cannot stop them from being generated. Therefore, to allow these toolkits to be used, service providers must be permitted to return any proprietary SOAP fault for any error condition.

To satisfy these criteria, a service provider should try to return a standard SOAP fault. When that is not possible a proprietary SOAP fault can be returned instead. The SOAP fault behaviour described in this document follows this logic: they should respond with some SOAP fault, a proprietary SOAP fault is allowed although it is preferred that a standard SOAP fault is used.

Service providers can also respond with a SOAP fault for other error conditions, including conditions internal or specific to a particular implementation or toolkit. That is, the error conditions identified in this document are not the complete set of all possible error conditions.

#### CONFIDENTIALITY AND PRIVACY

Service interface specifications need to ensure that the data sent in SOAP faults do not violate confidentiality and privacy. The nature of what is confidential and private depends on the service being specified, and is outside the scope of this document to define.

# 6 WS-Security security

## 6.1 PKI

### 6.1.1 Use of PKI

#### 6.1.1.1 Conformance criteria on service interface specifications

WS 6.1.1.1-1 Service interface specifications **MUST** use Public Key Infrastructure (PKI) as the security credentials for signing and encrypting with WS-Security (as described in section 6.2).

#### 6.1.1.2 Notes (non-normative)

The Public Key Infrastructure (PKI) is a standard for key management for asymmetric cryptographic algorithms.

Asymmetric cryptographic algorithms use two keys which are mathematically related. These keys are referred to as “key pairs”. This is different from symmetric cryptographic algorithms which use only one key.

One of the two keys is designated the “public key” and the other the “private key”. The private key is kept by the key owner and not disclosed to any other party. The public key can be widely disclosed to other parties.

Asymmetric cryptographic algorithms can be used for signing and encrypting data. For signing, the key owner uses their private key to produce the signature and anyone can validate the signature using the public key. If the signature is valid, it is proof that the signature creator had access to the private key.

For encrypting, the public key is used to encrypt the data. Only the corresponding private key can decrypt that encrypted data. Therefore, only the holder of that private key can access the data.

A public key certificate is a piece of data that contains a public key and other data. For example, the Subject Key Identifier is a value that is contained in the certificate. The certificate is digitally signed so it cannot be modified after it was issued.

In this document, the term “certificates” is always used to refer to a public key certificate. Be aware that in other contexts the term “certificate” can be misused to refer to the private key or to the pair of public and private keys.

It is beyond the scope of this document to define the policies relating to the issuing of PKI certificates. There will be authorities issuing PKI certificates, and NEHTA is working on identifying these authorities.

The use of PKI does not preclude the use of other security mechanisms to provide security features that PKI does not provide. For example, PKI does not preclude the use of Security Assertion Markup Language (SAML) for exchanging authentication and authorization information. These criteria do not describe whether SAML is used or not. If it is used, it will need to be specified by the service interface specification that uses it.

The criteria in this section only apply to PKI used in conjunction with WS-Security as defined in this document.

#### 6.1.1.3 Examples (non-normative)

##### POLICY EXAMPLE

The use of public key technology can be declared in a WSDL using the `AsymmetricBinding` assertion of `WS-SecurityPolicy`. This is shown in the example below.

**WSDL containing service instance information:**

```

<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    ...
    <sp:AsymmetricBinding>
    ...
    </sp:AsymmetricBinding>
  </wsp:Policy>
</definitions>

```

**6.1.2 Subject Key Identifier of X.509v3 certificates****6.1.2.1 Conformance criteria on SOAP message creators**

WS 6.1.2.1-1 SOAP message creators **MUST** use X.509v3 public key certificates which are uniquely identified by the Subject Key Identifier.

X.509v3 public key certificates are described in [X509-2005].

**6.1.2.2 Conformance criteria on service providers**

WS 6.1.2.2-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criterion WS 6.1.2.1-1.

WS 6.1.2.2-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `certificateSkiMissing`) if the SOAP request violates criterion WS 6.1.2.1-1.

**6.1.2.3 Notes (non-normative)**

The Subject Key Identifier is a value which is stored in an X.509v3 public key certificate. It is used to hold a value. This value is expected to be a 20 byte number.

This criterion is to ensure that there is always a mechanism to uniquely identify a certificate. This criterion does not prevent other mechanisms for identifying a certificate to also exist.

Since the Subject Key Identifier uniquely identifies the certificate, it can be used as a reference to the certificate. When the actual certificate is needed, a mechanism would be used to obtain the certificate that matches that unique Subject Key Identifier. This approach is used in the criteria of section 6.2.7 "Transmission of certificates".

**6.1.2.4 Examples (non-normative)****POLICY EXAMPLE**

This example shows how this criteria can be represented in the WSDL using WS-SecurityPolicy.

The `wss11` assertion allows WS-Security 1.1 options to be specified. One of these options is the `MustSupportRefKeyIdentifier` assertion. This assertion requires that SOAP message creators be able to generate key identifier

references and SOAP message consumers be able to process key identifier references.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    <sp:Wss11>
      <wsp:Policy>
        <sp:MustSupportRefKeyIdentifier/>
      </wsp:Policy>
    </sp:Wss11>
    ...
  </wsp:Policy>
</definitions>
```

### 6.1.3 Key usage

#### 6.1.3.1 Conformance criteria on SOAP message creators

WS 6.1.3.1-1 SOAP message creators **MUST** use X.509v3 certificates that either:

- Do not contain a *key usage* certificate extension, or
- Do contain a *key usage* certificate extension that **MUST** at least have both the *keyEncipherment* and *digitalSignature* bits set.

#### 6.1.3.2 Conformance criteria on service providers

WS 6.1.3.2-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates the criterion WS 6.1.3.1-1.

WS 6.1.3.2-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `certificateKeyUsage`) if the SOAP request violates criterion WS 6.1.3.1-1.

#### 6.1.3.3 Notes (non-normative)

A single key is used for both signing and encrypting operations. That is, the one key will be used to sign SOAP messages that entity produces, and the same key is used to decrypt SOAP messages that other entities encrypt for that party.

Dual keys, where one key pair is used for encrypting data and a different key pair is used for digital signing, are not permitted by these criteria.

These criteria are present because some Web services toolkits are unable to use dual keys.

A description of key usage can be found in section 4.2.1.3 of [RFC3280] or in section 8.2.2.3 of [X509-2005].

## 6.1.4 Certificates with identifiers

### 6.1.4.1 Conformance criteria on SOAP message creators

WS 6.1.4.1-1 SOAP message creators **MUST** use X.509v3 certificates that can be associated with the identity of the service provider or service requestor.

### 6.1.4.2 Conformance criteria on service providers

WS 6.1.4.2-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criterion WS 6.1.4.1-1.

WS 6.1.4.2-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `certificateUnidentified`) if the SOAP request violates criterion WS 6.1.4.1-1.

### 6.1.4.3 Notes (non-normative)

A unique identifier for the entity is necessary to ensure the correct relationship between the message, the service sending or receiving the message, and the organisation operating the service. If this relationship cannot be established, there is a high security risk for fraud and impersonation attacks.

These criteria ensure that there is always a mechanism to uniquely associate a certificate to its owner, that this mechanism will be present, and that all parties know how and when to use it.

## 6.2 WS-Security

### 6.2.1 WS-Security 1.1

#### 6.2.1.1 Conformance criteria on service interface specifications

WS 6.2.1.1-1 Service interface specifications **MUST** specify the use of WS-Security 1.1 to sign and encrypt SOAP messages.

WS-Security 1.1 is defined by [WSS2006].

#### 6.2.1.2 Notes (non-normative)

WS-Security defines mechanisms for preserving the confidentiality and integrity of messages through the use of encryption and digital signatures. It provides a standard way of using XML Encryption to encrypt selected parts of the message [XENC2002], and it provides a standard way of using XML Digital Signatures to associate a digital signature with parts of the message [DSIG2002]. It also provides a mechanism for sending security tokens as a part of the message.

WS-Security provides end-to-end security, which is important when messages are sent across multiple hops. The security must be maintained between the sender and the ultimate receiver.

## 6.2.2 WS-Security timestamp

### 6.2.2.1 Conformance criteria on service interface specifications

- WS 6.2.2.1-1 Service interface specifications **MUST** define a policy on the interpretation of WS-Security timestamps. That policy is to include specifying the values identified in WS 6.2.2.1-2, WS 6.2.2.1-3 and WS 6.2.2.1-4.<sup>1</sup>
- WS 6.2.2.1-2 Service interface specifications **MUST** define a value for the *maximum clock skew allowance*, being the maximum difference that a computer's clock can have from actual UTC time.
- WS 6.2.2.1-3 Service interface specifications **MUST** define a value for the *maximum transit delay*, being the maximum delay from when the SOAP message starts sending from the SOAP message creator to when the SOAP message consumer starts receiving it. This delay is defined in the absence of any clock skew; that is, it assumes both the sender and consumer's clocks are synchronised.
- WS 6.2.2.1-4 Service interface specifications **SHOULD** specify the expected maximum time it takes for SOAP message to be transmitted. This is the time measured from when the consumer starts receiving data to when it finishes receiving the SOAP message.

### 6.2.2.2 Conformance criteria on SOAP message creators

#### CREATED TIMESTAMP

- WS 6.2.2.2-1 SOAP message creators **MUST** include a `Created` timestamp in the WS-Security timestamp element.
- WS 6.2.2.2-2 SOAP message creators **MUST** represent the `Created` timestamp using the XML Schema `xsd:dateTime` datatype.
- WS 6.2.2.2-3 SOAP message creators **MUST** express the `Created` timestamp with an explicit timezone of UTC.
- WS 6.2.2.2-4 SOAP message creators **MUST** ensure that the `Created` timestamp is within the *maximum clock skew allowance* of the actual time the SOAP message started to be sent.
- WS 6.2.2.2-5 SOAP message creators **MUST** behave correctly when the `Created` timestamp is ignored by the SOAP message consumer.

#### EXPIRES TIME

- WS 6.2.2.2-6 SOAP message creators **MAY** include an `Expires` time in the WS-Security timestamp element.
- WS 6.2.2.2-7 SOAP message creators **MUST** represent any `Expires` time using the XML Schema `xsd:dateTime` datatype.
- WS 6.2.2.2-8 SOAP message creators **MUST** express any `Expires` time with an explicit timezone of UTC.
- WS 6.2.2.2-9 SOAP message creators **MUST** ensure that the `Expires` time can be treated as a value that is relative to the `Created` timestamp. That is, they are both measured from the same clock.

<sup>1</sup> The policy must not conflict with the criteria in this profile. For example, the policy cannot declare that the `Created` timestamp is not present, because that would violate criterion WS 6.2.2.2-1.

- WS 6.2.2.2-10 SOAP message creators **MUST** behave the same when the `Expires` timestamp is ignored by the SOAP message consumer as when the `Expires` timestamp is processed by the SOAP message consumer and the `Expires` timestamp has been exceeded.

### 6.2.2.3 Conformance criteria on SOAP message consumers

#### CREATED TIMESTAMP

- WS 6.2.2.3-1 SOAP message consumers **MAY** ignore the `Created` timestamp and not expire SOAP messages based on it.
- WS 6.2.2.3-2 SOAP message consumers **MAY** check if SOAP messages have expired based on the `Created` timestamp.

If a SOAP message consumer chooses to check if the SOAP message has expired based on the `Created` timestamp, it needs to conform to the following two criteria WS 6.2.2.3-3 and WS 6.2.2.3-4:

- WS 6.2.2.3-3 SOAP message consumers **MUST NOT** reject a SOAP message that has taken between zero and the *maximum transit delay* to start to arrive.
- WS 6.2.2.3-4 SOAP message consumers **SHOULD** reject a SOAP message that has taken less than zero or more than the *maximum transit delay* to start to arrive.

#### EXPIRES TIME

- WS 6.2.2.3-5 SOAP message consumers **MAY** ignore the `Expires` time and not expire SOAP messages based on it.
- WS 6.2.2.3-6 SOAP message consumers **MAY** check if SOAP messages have expired based on the `Expires` time.

If a SOAP message consumer chooses to check if a SOAP message has expired based on the `Expires` time, it needs to conform to the following criterion WS 6.2.2.3-7:

- WS 6.2.2.3-7 SOAP message consumers **SHOULD** reject a SOAP message if the SOAP message started to arrive after the `Expires` time in the SOAP message.

### 6.2.2.4 Conformance criteria on service providers

- WS 6.2.2.4-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request is missing the `Created` timestamp or it is of the wrong format.
- WS 6.2.2.4-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badTimestamp`) if the SOAP request is missing the `Created` timestamp or it is of the wrong format.
- WS 6.2.2.4-3 Service providers **SHOULD** respond with a SOAP fault if the SOAP request message was rejected because it has expired.
- WS 6.2.2.4-4 Service providers **SHOULD** respond with a WS-Security `wsu:MessageExpired` SOAP fault if the SOAP request message was rejected because it has expired.

### 6.2.2.5 Notes (non-normative)

Time is important and needs to be handled in a correct and interoperable manner.

Dealing with time in a distributed environment is complicated. The following sections will describe some of the issues that are important to enable systems to successfully integrate together.

When implementing these criteria, it is important to recognise that they have been designed to allow for flexibility in how they are satisfied.

For example, consider if a criterion says that a message must be accepted if the timestamp is younger than 10 minutes and another criterion says it should be rejected if it is older than 10 minutes. The implementation can accept messages younger than, say, 12 minutes and reject messages older than 12 minutes. This is because the second criteria is a “should” and not a “must,” so there is flexibility in how it is satisfied.

#### CLOCK SKEW

In a distributed system, clocks on different machines cannot be guaranteed to be the same. The concept of clock skew is introduced to account for the discrepancies in the clocks.

The values chosen for the maximum clock skew allowance determine how accurate the computer clocks must be.

These criteria make it the responsibility of the service interface specification to specify what the maximum allowed clock skew is. This is because the appropriate value would depend on the service’s accuracy requirements and on the ability to ensure that the computers on which the service instances are deployed keep their clocks accurate—both of which are outside the scope of this document.

A small maximum clock skew means the timestamps could be relied upon to be more accurate, but places a stronger demand on the deployed computers to keep their clocks accurate. A large maximum clock skew means the timestamps are less accurate, but places less of a demand on the deployed computers to keep their clocks accurate. If the maximum clock skew is infinite, the timestamps cannot be relied upon to be accurate and there is no obligations placed on the deployed computers to keep their clocks accurate.

It is recommended that a clock synchronisation protocol such as Network Time Protocol (NTP) be used to keep clocks synchronised with a trusted time source. Although highly recommended, it is outside the scope of this document to mandate the use of NTP. That is something the service interface specification policy might define.

Even if time synchronisation is mandated, it does not prevent malicious parties from deliberately using incorrect clocks. Therefore, timestamps must always be treated as a guide and not as an established fact.

The definition of clock skew used here is usually different from the definition used by Web service toolkits. This definition compares the clock to the UTC time. Some toolkits define it as comparing the clocks on the two computers, in which case that value would be twice the value of the clock skew as defined in this document.

Clock skew is defined differently in this document so that single service instances can be tested for conformance. The toolkit definition depends on the clock values of all external computers that could interact with the service instance, and that is impractical to test.

#### SECURITY CAUTIONS

Caution should be taken when processing SOAP messages, because malicious parties might:

- Change their clocks to provide a false timestamp.
- Replay captured SOAP messages.

- Modifying the `Expires` time to get around some security checks. For example, setting it to a value much larger than the maximum transit delay.

#### WS-SECURITY TIMESTAMPS

Timestamps are optional in WS-Security [WSS2006]. However, some Web service toolkits require creation timestamps to be present in the messages they receive. Therefore, to ensure compatibility with those toolkits `Created` timestamps must be included in all messages.

The `Created` timestamp will be a time that relates to the transportation of the SOAP message. It should not be used to represent times related to the business process of the message: those application timestamps should be included in the body of the SOAP message.

#### MAXIMUM TRANSIT DELAY

WS-Security does not specify when a SOAP message is deemed to have been received. As a result, there is no standard interpretation of the timestamps. Some toolkits assume it is when the first bit of the SOAP message is received, while others assume it is when the last bit is received. When the SOAP messages are small and the network is fast, this difference is small and inconsequential. However, if the SOAP messages are large and/or the network is slow, the difference can be large and can cause messages to be incorrectly expired or accepted.

The value of the maximum transit delay is defined here as the time between the sending of the first bit by the SOAP message creator and the receiving of the first bit at the SOAP message consumer. This is so that its value will only depend on network latency. If it is calculated using the time of receiving the last bit at the SOAP message consumer, then the size of the SOAP message and the network bandwidth would have to be taken into account—making it difficult to define a maximum value when those factors are variable and/or unknown.

Be aware that some Web service toolkits use a different definition of when a message is received. If so, the service provider must allow for this difference in order to produce the behaviour defined in the above criteria. For example, if it uses the time the last bit is received, it might have to increase its transit delay to accommodate the largest expected SOAP message and the slowest network. This will cause smaller messages to slip through the expiry checking, but that is acceptable under these guidelines because SOAP message consumers can choose to not detect expired messages. This is why the criteria on rejecting messages are specified as “should” instead of “must”.

Criterion WS 6.2.2.1-4 is designed to provide an upper bound to compensation needed to account for the difference in when messages are considered to have been received. This message transmission time needs to take into account the maximum message size and the minimum bandwidth that could be available.

The effect of clock skew needs to be taken into account when dealing with transit delay times. The SOAP message consumer can be certain that the maximum transit delay has been exceeded when the transit time has been exceeded by more than the combined clock skew of both parties. The SOAP message consumer can be certain that the maximum transit delay has not been exceeded when the transit time is under by a margin more than the combined clock skew of both parties. In between those two situations, the SOAP message consumer can assume the maximum transit delay has not been exceeded. This satisfies the criteria because it “should” detect the error, not it “must” detect the error.

When specifying a maximum transit delay, its impact on implementing other criteria needs to be considered. Many toolkits provide SOAP message duplication detection by maintaining a buffer of previously received messages.

The size of this buffer is usually related to the maximum transit delay. A large maximum transit delay would require a large buffer to be maintained. An infinite maximum transit delay could result in SOAP message duplicate detection to be impossible to implement.

#### DIGITAL SIGNING

In keeping with section 6.2.3, the WS-Security `timestamp` element must be digitally signed.

#### 6.2.2.6 Examples (non-normative)

##### POLICY EXAMPLE

These criteria can be met by using the `IncludeTimestamp` assertion in `WS-SecurityPolicy`. This assertion requires WS-Security timestamps to be included and digitally signed in SOAP messages [WSSPL2007]. The following example shows how to declare the `IncludeTimestamp` assertion in a WSDL specification.

##### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    ...
    <sp:AsymmetricBinding>
      <wsp:Policy>
        ...
        <sp:IncludeTimestamp/>
        ...
      </wsp:Policy>
    </sp:AsymmetricBinding>
  </wsp:Policy>
</definitions>
```

##### SOAP MESSAGE EXAMPLE

This example shows a WS-Security timestamp in a SOAP message.

##### SOAP Message:

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <soap:Header>

    <wsse:Security>
      <wsu:Timestamp>
        <wsu:Created>2007-01-26T02:55:38Z</wsu:Created>
        <wsu:Expires>2007-01-26T03:00:38Z</wsu:Expires>
      </wsu:Timestamp>
      ...
    </wsse:Security>

  </soap:Header>
  <soap:Body> ... </soap:Body>
</soap:Envelope>
```

## 6.2.3 Digital signatures

### 6.2.3.1 Obligation on SOAP message creators

- WS 6.2.3.1-1 SOAP message creators **MUST** digitally sign the SOAP body, the WS-Security timestamp and all SOAP headers (not including the WS-Security header itself).
- WS 6.2.3.1-2 SOAP message creators **MUST** calculate digital signatures over the entire element (i.e. including the start and end tags of the element, not just the contents of the element).
- WS 6.2.3.1-3 SOAP message creators **MAY** have the `Reference` elements in any order inside the `SignedInfo` element.

### 6.2.3.2 Conformance criteria on SOAP message consumers

- WS 6.2.3.2-1 SOAP message consumers **MUST** reject SOAP messages which do not have their body and expected headers digitally signed.
- WS 6.2.3.2-2 SOAP message consumers **MUST** validate that the digital signature is correct and matches the SOAP body and headers before using the data.
- WS 6.2.3.2-3 SOAP message consumers **MUST NOT** use data in any part of the message which has not been signed.
- WS 6.2.3.2-4 SOAP message consumers **MUST NOT** assume that items are signed in any particular order.
- WS 6.2.3.2-5 SOAP message consumers **MUST NOT** assume that the sender of the message is the party that signed it.

### 6.2.3.3 Conformance criteria on service providers

- WS 6.2.3.3-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criteria WS 6.2.3.1-1, WS 6.2.3.1-2, or WS 6.2.3.1-3.
- WS 6.2.3.3-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badSignature`) if the SOAP request violates criteria WS 6.2.3.1-1, WS 6.2.3.1-2, or WS 6.2.3.1-3.
- WS 6.2.3.3-3 Service providers **MUST** sign SOAP responses using the key associated with the certificate that was used to encrypt the corresponding SOAP request.

### 6.2.3.4 Notes (non-normative)

This digital signature can be used to:

- Validate the integrity of the data;
- Identify the signing party; and
- Authenticate the signing party.

It is important to recognise that the signature does *not* identify nor prove who created the SOAP message. The creator of the SOAP message is not necessarily the same as who created the signature. See section 6.2.5.3 for a detailed discussion of this issue.

The signature should be created by the initial sender of the SOAP message.

These criteria only cover the transportation of the message, and not signatures relating to business functionality. If signatures are required relating to business functionality of the message content, then they should be

incorporated into the body of the message. The signature on the SOAP message only relates to identifying the entity that sent the message.

All data in the message must be signed, because any unsigned data could be tampered with and poses a security risk. This means the SOAP body and all headers are signed. The only exception is the security header, in which only the timestamp portion must be signed.

These criteria state that the entire SOAP body is signed. Although the XML Signature mechanism allows signing only specific parts of an XML document, this is not used here because the purpose of this signature is to ensure the integrity of the entire SOAP body.

Different toolkits can generate `SignedInfo` elements with the `Reference` elements inside it in different order. For example, some may put the `Reference` element referring to the body before the `Reference` element referring to the WS-Security timestamp, and some the other way around. Therefore, it is important for SOAP message consumers to not expect signatures containing the `Reference` elements in any particular order. They need to accept signatures that have `Reference` elements in any order, as long as the other aspects of the signature are correct.

There can be other headers in the SOAP message that are not signed. They may be inserted into the SOAP message by other parties. These criteria allow them to be ignored—whether they are signed or not. If the data in them is used, then they have to be signed.

### 6.2.3.5 Examples (non-normative)

#### POLICY EXAMPLE

This example shows how to represent these criteria using WS-SecurityPolicy.

The `SignedParts` assertion specifies which parts of the SOAP messages need to be signed. If the `SignedParts` assertion has no child elements, then it means that the body and all headers to the ultimate receiver must be signed.

The `IncludeTimestamp` assertion requires timestamps to be signed when they are included in messages.

The `OnlySignEntireHeadersAndBody` assertion states that the entire element should be signed, not the content.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
  ...
</binding>
...
<wsp:Policy wsu:Id="SecurityPolicy">
  ...
  <sp:SignedParts/>
  ...
  <sp:AsymmetricBinding>
    <wsp:Policy>
      ...
      <sp:IncludeTimestamp/>
      <sp:OnlySignEntireHeadersAndBody/>
      ...
    </wsp:Policy>
  </sp:AsymmetricBinding>
</wsp:Policy>
</definitions>
```

## 6.2.4 Encryption

### 6.2.4.1 Obligation on service interface specifications

WS 6.2.4.1-1 Service interface specifications **MUST NOT** put confidential data in the SOAP headers. Confidential data can only go into the SOAP body, which will be encrypted.

### 6.2.4.2 Obligation on SOAP message creators

WS 6.2.4.2-1 SOAP message creators **MUST** encrypt the entire SOAP body.

WS 6.2.4.2-2 SOAP message creators **MUST** encrypt the WS-Security message signature header.

WS 6.2.4.2-3 SOAP message creators **SHOULD NOT** encrypt any other SOAP header (i.e. any header other than the WS-Security message signature header).

### 6.2.4.3 Conformance criteria on SOAP message consumers

WS 6.2.4.3-1 SOAP message consumers **MUST** reject SOAP messages which do not have the entire SOAP body encrypted.

WS 6.2.4.3-2 SOAP message consumers **MUST** reject SOAP messages which do not have the WS-Security message signature header encrypted.

### 6.2.4.4 Conformance criteria on service providers

WS 6.2.4.4-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criteria WS 6.2.4.2-1, WS 6.2.4.2-2, or WS 6.2.4.2-3.

WS 6.2.4.4-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badEncryption`) if the SOAP request violates criteria WS 6.2.4.2-1, WS 6.2.4.2-2, or WS 6.2.4.2-3.

WS 6.2.4.4-3 Service providers **MUST** encrypt SOAP responses using the service consumer's certificate (i.e. the one associated with the key that was used to sign the corresponding SOAP request).

### 6.2.4.5 Notes (non-normative)

The confidentiality of the data in the SOAP message needs to be preserved, so that third parties cannot access the data.

What data is confidential will depend on the particular service being specified, and is outside the scope of this document to define.

The entire SOAP body is encrypted as a single block. Although XML Encryption allows portions of the XML document to be encrypted, this feature is not used here to encrypt a portion of the SOAP body. Specifying that the entire SOAP body is encrypted is simpler to understand and implement properly—reducing the risk that sensitive data is not encrypted. Encrypting portions of the SOAP body is currently not well supported by toolkits.

The headers (other than the header containing the message signature) usually contain data that is used by the toolkits or intermediaries, so they should not be encrypted otherwise those parties would not be able to access them.

The SOAP message signature is encrypted because this can slightly reduce the (small) risk of the message being tampered with

It should be noted that the correct way to apply XML Encryption to a SOAP message is to use:

- Element encryption on the signature header. This ensures that the element start and end tags are also encrypted.
- Element content encryption on the SOAP body. This ensures that the SOAP body start and end tags are not encrypted. This is necessary because the message must still be a valid SOAP message after encryption has been applied to it.

Toolkits can be very strict about which elements are encrypted and which are not. This is why these criteria suggest that other headers should not be encrypted. If additional headers need to be encrypted, it must be well documented as a part of the service interface.

The SOAP message is encrypted for the intended service provider. There is no delegation model at the SOAP message level, where one service provider can delegate another to act on its behalf. Delegation needs to be implemented at a higher level of the protocol.

Service providers can encrypt SOAP responses using the service consumer's certificate because that certificate is sent to the service provider in the SOAP request. It is the same certificate that the service consumer used to sign the SOAP request. It can be used for encrypting the SOAP response because these criteria apply in conjunction with the criteria in section 6.1.3, which says that these certificates are suitable for both signing and encrypting.

#### 6.2.4.6 Example (non-normative)

POLICY EXAMPLE

This example shows how these criteria can be represented in a WSDL specification.

The `EncryptedParts` assertion specifies which parts of the SOAP messages need to be encrypted. If the `EncryptedParts` assertion has no child elements, then it means that the SOAP body must be encrypted.

The `EncryptSignature` assertion specifies that the WS-Security signature must be encrypted.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    ...
    <sp:EncryptedParts/>
    <sp:AsymmetricBinding>
      <wsp:Policy>
        ...
        <sp:EncryptSignature/>
        ...
      </wsp:Policy>
    </sp:AsymmetricBinding>
  </wsp:Policy>
</definitions>
```

## 6.2.5 Sign before encryption

### 6.2.5.1 Obligation on SOAP message creators

WS 6.2.5.1-1 SOAP message creators **MUST** digitally sign before encrypting.

### 6.2.5.2 Obligation on SOAP message consumers

WS 6.2.5.2-1 SOAP message consumers **MUST** reject messages that have not been signed before being encrypted.

WS 6.2.5.2-2 SOAP message consumers **MUST** follow the Lax "Security Header Layout" property as defined in WS-Security Policy [WSSPL2007].

### 6.2.5.3 Conformance criteria on service providers

WS 6.2.5.3-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criterion WS 6.2.5.1-1.

WS 6.2.5.3-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badSigEncOrder`) if the SOAP request violates criterion WS 6.2.5.1-1.

### 6.2.5.4 Notes (non-normative)

The order in which signing and encryption occurs matters. When messages are encrypted before being signed, the signature is computed over the cipher text. When messages are signed before encryption, the signature is computed over the plain text. The order must be agreed upon.

Sign-before-encrypt was chosen because the integrity checking advantages of the other approach, encrypt-before-sign, are less important. A possible reason for choosing encrypt-before-sign is that the message's integrity can be checked first, before processing time is spent on decrypting the data. If the data has been corrupted, decryption does not need to happen. This reason is less important if the protocol used to deliver the SOAP messages have their own integrity checks (as is the case with TCP, which HTTP uses).

The use of sign-before-encrypt helps prevent man-in-the-middle attacks where an attacker listens in on requests from a service invoker. If the opposite encrypt-before-sign approach is used, the attacker can copy the encrypted data from a request, create a new SOAP request and sign it with their own key. Assuming that their key is recognised by the service provider, it would process the false request as a legitimate request. Although the attacker cannot access the data in the request, it can access the response and from it possibly infer what was in the request. Thus, attackers only need to obtain one set of keys in order to compromise the messages for everyone in the system. This attack is not possible with sign-before-encrypting since there is no way to change the signature because it is encrypted.

The sign-before-encrypt approach may also have benefits in auditing, if the receiver already stores the received data in an unaltered form. It would only need to store the signature in an audit trail because it corresponds to the data that is already being stored—therefore saving storage space. However, if encrypt-before-sign was used, it would need to store the decrypted data, the signature and the encrypted data in the audit trail.

Implementations need to be aware of the limitations of sign-before-encrypt. The signature, by itself, does not indicate who sent the message—only who created the message. The creator A signs the message, encrypts it for party B and sends it to party B. Party B decrypts the message, keeps the signature, and encrypts it for party C and sends it to party C. Party C must not incorrectly assume that party A sent them the message, even though it was

signed by A. These limitations of sign-before-encrypt are discussed in further detail in [DAVI2001a] [DAVI2001b].

This above limitation can be solved by including the author's intended recipient identifier in the data that is signed. This information can be included in the payloads. In this example, C will see that A only intended to send the message to B.

#### LAX ORDER OF TIMESTAMPS

The Lax "security Header Layout" policy in WS-Security Policy specifies that the items are added to the security header in any order that conforms to Web Services Security [WSS2006]. That is, it cannot reject messages because of the location of the timestamp within the WS-Security header.

This is necessary because in the WS-Security header, the position of the timestamp may vary between different toolkits. Some toolkits control the order of signing and encrypting through a mechanism that also determines where the timestamp is placed in the security header. Other toolkits do not allow the position of the timestamp to be changed. Therefore, SOAP message consumers must be able to accept messages regardless of where the timestamp appears.

#### 6.2.5.5 Examples (non-normative)

##### POLICY EXAMPLE

The following example shows how to represent these criteria using WS-SecurityPolicy in a WSDL. The `Protection Order` property of WS-SecurityPolicy is by default set to signing before encrypting. Thus, no explicit assertion needs to be specified. The `Lax` layout assertion specifies that the order of items in the security header only have to conform to the rules of WS-Security; the WS-Security timestamp does not have to appear in particular location (e.g. the first or last item) in the security header.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>

<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    ...
    <sp:AsymmetricBinding>
      <wsp:Policy>
        ...
        <sp:Layout>
          <wsp:Policy>
            <sp:Lax/>
          </wsp:Policy>
        </sp:Layout>
        ...
      </wsp:Policy>
    </sp:AsymmetricBinding>
  </wsp:Policy>
</definitions>
```

## 6.2.6 Algorithms

### 6.2.6.1 Obligation on SOAP message creators

- WS 6.2.6.1-1 SOAP message creators **MUST** use the cryptographic algorithms from the Basic256Rsa15 algorithm suite for signing and encrypting SOAP messages. The Basic256Rsa15 algorithm suite is defined in WS-SecurityPolicy 1.2 [WSSPL2007].

### 6.2.6.2 Conformance criteria on SOAP message consumers

- WS 6.2.6.2-1 SOAP message consumers **MUST** reject SOAP messages which do not use cryptographic algorithms from the Basic256Rsa15 algorithm suite.

### 6.2.6.3 Conformance criteria on service providers

- WS 6.2.6.3-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criterion WS 6.2.6.1-1.
- WS 6.2.6.3-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an appropriate error code from the following list) if the SOAP request violates criterion WS 6.2.6.1-1:
- `badAlgorithmDataEncryption`
  - `badAlgorithmKeyEncryption`
  - `badAlgorithmC14N`
  - `badAlgorithmDigest`
  - `badAlgorithmSignature`

### 6.2.6.4 Notes (non-normative)

In summary, the Basic256Rsa15 algorithms are:

- Encrypt data using the symmetric AES-256 algorithm;
- Encrypt the symmetric key that was used to encrypt data using the RSA 1.5 algorithm;
- Transform data using Exclusive XML Canonicalization before calculating digest values;
- Calculate digest values using the SHA-1 algorithm; and
- Sign digest values using the RSA SHA-1 algorithm.

Different cryptographic algorithms can be used in WS-Security. However, if the SOAP message creator uses an algorithm that the SOAP message consumer doesn't understand then they will be incompatible.

The RSA public key algorithms (i.e. RSA SHA-1 and RSA 1.5), SHA hashing algorithms (i.e. SHA-1), and AES symmetric encryption algorithms (i.e. AES-256) are approved by the Australian Government Defence Signals Directorate [ACSI33].

The algorithms in the Basic256Rsa15 suite are identified by the URLs in this table:

Algorithm	Name	Identifier
Data encryption	AES-256	<a href="http://www.w3.org/2001/04/xmlenc#aes256-cbc">http://www.w3.org/2001/04/xmlenc#aes256-cbc</a>
Key encryption	RSA 1.5	<a href="http://www.w3.org/2001/04/xmlenc#rsa-1_5">http://www.w3.org/2001/04/xmlenc#rsa-1_5</a>

Digest	SHA-1	http://www.w3.org/2000/09/xmldsig#sha1
Signature	RSA SHA-1	http://www.w3.org/2000/09/xmldsig#rsa-sha1
Canonicalization	Exclusive XML	http://www.w3.org/2001/10/xml-exc-c14n#
Transformations	None (other than Exclusive XML canonicalization)	

Note: RSA SHA-1 and the Exclusive XML Canonicalization algorithms come from WS-SecurityPolicy's base algorithm suite, which is used by WS-SecurityPolicy's Basic256Rsa15 algorithm suite.

### 6.2.6.5 Examples (non-normative)

#### POLICY EXAMPLE

This example shows how to indicate the use of the Basic256Rsa15 algorithm suite in the WSDL.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    ...
    <sp:AsymmetricBinding>
      <wsp:Policy>
        ...
        <sp:AlgorithmSuite>
          <wsp:Policy>
            <sp:Basic256Rsa15/>
          </wsp:Policy>
        </sp:AlgorithmSuite>
        ...
      </wsp:Policy>
    </sp:AsymmetricBinding>
  </wsp:Policy>
</definitions>
```

#### SOAP MESSAGE EXAMPLE

This example shows a SOAP request message that uses the cryptographic algorithms described in these criteria. It shows the signature block in plain text (i.e. not encrypted) so that it can be read. However, in a correct SOAP message that follows section 6.2.3.5, the signature block would be encrypted.

#### SOAP Message:

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <soap:Header>
    <wsse:Security>
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        ...
      </xenc:EncryptedKey>
```

```

...
<ds:Signature>
  <!-- This element should actually be encrypted -->
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference ...>
      <ds:Transforms>
        <ds:Transform
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        </ds:Transforms>
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      ...
    </ds:Reference>
  ...
</ds:Signature>
...
</wsse:Security>
</soap:Header>

<soap:Body>
  <xenc:EncryptedData ...>
    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
    ...
  </xenc:EncryptedData>
</soap:Body>
</soap:Envelope>

```

## 6.2.7 Transmission of certificates

### 6.2.7.1 Conformance criteria on service invokers

- WS 6.2.7.1-1 Service invokers **MUST** include the signing certificate in requests using the "Direct Reference" mechanism of WS-Security 1.1. The signing certificate is the one that corresponds to the private key used to sign the SOAP request.
- WS 6.2.7.1-2 Service invokers **MUST** reference the service providers' certificate using the Subject Key Identifier in requests. This certificate is the one used to encrypt the SOAP request.

### 6.2.7.2 Conformance criteria on service providers

- WS 6.2.7.2-1 Service providers **MUST** reference their certificate using the Subject Key Identifier in responses. This certificate corresponds to the private key used to sign the SOAP response.
- WS 6.2.7.2-2 Service providers **MUST** reference the service invokers' certificate using the Subject Key Identifier in responses. This certificate is the one used to encrypt the SOAP response.
- WS 6.2.7.2-3 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criteria WS 6.2.7.1-1 or WS 6.2.7.1-2.
- WS 6.2.7.2-4 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badCertificateTransmit`) if the SOAP request violates criteria WS 6.2.7.1-1 or WS 6.2.7.1-2.

### 6.2.7.3 Conformance criteria on service interface specifications

- WS 6.2.7.3-1 Service interface specifications **MUST** identify or define the rules for verifying certificates.

#### 6.2.7.4 Notes (non-normative)

WS-Security provides a number of different options for describing the certificates being used for signing and encryption in the SOAP message. SOAP message consumers and creators need to use compatible approaches to ensure interoperability. These criteria address this problem. The criteria are written in terms of service invokers and service providers, and those criteria also have Conformance criteria on SOAP message consumers and creators (as described in section 1.3.7 "Artefacts").

These criteria define the mechanisms used in the requests and responses because some toolkits are very strict in what they expect to process. This applies especially to toolkits whose security configuration is based on the WS-SecurityPolicy standard [WSSPL2007]. For example, if the recipient token inclusion policy is set to "Never," then requests where the encryption certificate is included in the message will be rejected.

The two mechanisms used here are:

- Direct Reference used with a copy of the full X.509 certificate included in the SOAP message. The certificate is placed with the SOAP message, so the recipient does not need to lookup any external certificate directory to obtain it<sup>2</sup>.
- Reference to a Subject Key Identifier. The Subject Key Identifier field from the X.509 certificate is placed in the SOAP message. The recipient will have to obtain the certificate from an external certificate directory. The Subject Key Identifier is a unique identifier for the X.509 certificate.

The Direct Reference and Subject Key Identifier mechanisms have been chosen because they are commonly supported by toolkits. Although WS-Security and the X.509 Certificate Token Profile [XCTP2006] describe other mechanisms, they are not well supported by toolkits.

The Subject Key Identifier mechanism is generally preferred because in most cases there is no need to include the certificate in the message. One case where the inclusion of the certificate has some advantages is having the service invoker's signing certificate in the request message, which is why the "Direct Reference" mechanism is used for this certificate. For instance, the service provider can validate a request message's digital signature without having to retrieve the service invoker's certificate (assuming they trust the root certificate authority of the certificate). Also, the service provider can use this certificate to encrypt the response. In both cases, the service provider does not have to rely upon an external certificate directory.

The service provider does not necessarily have to use the service invoker's certificate that was included in the SOAP message: it is allowed to retrieve it from an alternative trusted source.

The table below summarises the criteria in this section:

	<b>Signature Key</b>	<b>Encryption Key</b>
<b>Request SOAP message</b>	Direct reference (Service invoker's certificate)	Reference the Subject Key Identifier (Service provider's certificate)
<b>Response SOAP message</b>	Reference the Subject Key Identifier (Service provider's certificate)	Reference the Subject Key Identifier (Service invoker's certificate)

<sup>2</sup> This section uses the terminology from the WS-Security 1.1 specification. Even though the mechanism is called "Direct Reference" it is not being used to "reference" the certificate. The term means to implant or embed the actual certificate in the SOAP message. However, the word "embedded" is not used here because it is the name of another mechanism in WS-Security.

If section 6.1.2.4 is followed, then each party only has a single key pair. The service invoker's signature key is the same as their encryption key and the service provider's signature key is the same as their encryption key.

### 6.2.7.5 Example (non-normative)

#### POLICY EXAMPLE

This example shows how to declare these criteria in a WSDL specification using WS-SecurityPolicy.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  ...>
  ...
  <binding ...>
    <wsp:PolicyReference URI="#SecurityPolicy"/>
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="SecurityPolicy">
    ...
    <sp:AsymmetricBinding>
      <wsp:Policy>
        ...
        <sp:InitiatorToken>
          <wsp:Policy>
            <sp:X509Token
              sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
              </sp:X509Token>
            </wsp:Policy>
          </sp:InitiatorToken>
          <sp:RecipientToken>
            <wsp:Policy>
              <sp:X509Token
                sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/Never">
                </sp:X509Token>
              </wsp:Policy>
            </sp:RecipientToken>
            ...
          </wsp:Policy>
        </sp:AsymmetricBinding>
      </wsp:Policy>
    </definitions>
```

#### SOAP MESSAGE EXAMPLE

This example shows how certificates are handled in SOAP messages.

This SOAP request message contains (in order):

- Reference to the Subject Key Identifier for the encrypting certificate of the service provider (i.e. the consumer of this SOAP message);
- Direct Reference (i.e. implanted copy) to the signing certificate of the service invoker (i.e. the creator of this SOAP message);
- Signature block for the message, which indicates that the signer's certificate is the one implanted in this SOAP message.

This example shows the signature block in plain text (i.e. not encrypted) so that it can be read. However, in a correct SOAP message that follows section 6.2.3.5, the signature block would be encrypted.

#### SOAP Request:

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
```

```

    xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    utility-1.0.xsd"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    <soap:Header>
      <wss:Security>
        <!-- Service provider's encryption certificate referenced using SKI -->
        <xenc:EncryptedKey>
          ...
          <ds:KeyInfo>
            <wss:SecurityTokenReference>
              <wss:KeyIdentifier>
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
                wss-soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                x509-token-profile-1.0#X509SubjectKeyIdentifier"> ...
              </wss:KeyIdentifier>
            </wss:SecurityTokenReference>
          </ds:KeyInfo>
        </xenc:EncryptedKey>

        <!-- Service invoker's signing certificate included in this message -->
        <wss:BinarySecurityToken
          wsu:Id="123"
          EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
          soap-message-security-1.0#Base64Binary"
          ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
          token-profile-1.0#X509v3"> ...
        </wss:BinarySecurityToken>
        <!-- Signature block, which references the
          service invoker's signing certificate given above -->
        <ds:Signature>
          <!-- This element should actually be encrypted -->
          ...
          <ds:KeyInfo>
            <wss:SecurityTokenReference>
              <wss:Reference>
                URI="#123"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                x509-token-profile-1.0#X509v3"/>
              </wss:SecurityTokenReference>
            </ds:KeyInfo>
          </ds:Signature>
          ...
        </wss:Security>
      </soap:Header>
      <soap:Body>
        <!-- This is encrypted data -->
        ...
      </soap:Body>
    </soap:Envelope>

```

This SOAP response message contains (in order):

- The Subject Key Identifier for the encrypting certificate of the service invoker (i.e. the consumer of this SOAP response message);
- Signature block for the message. Inside this block is a reference to the Subject Key Identifier the signing certificate of the service provider (i.e. the creator of this SOAP message).

#### SOAP Response:

```

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  utility-1.0.xsd"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  <soap:Header>
    <wss:Security>
      <!-- Service invoker's encryption certificate referenced using SKI -->
      <xenc:EncryptedKey>
        <ds:KeyInfo>
          <wss:SecurityTokenReference>
            <wss:KeyIdentifier>

```

```
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-  
wss-soap-message-security-1.0#Base64Binary"  
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-  
x509-token-profile-1.0#X509SubjectKeyIdentifier">  
        ... </wsse:KeyIdentifier>  
    </wsse:SecurityTokenReference>  
</ds:KeyInfo>  
</xenc:EncryptedKey>  
  
<!-- Signature block, which references the  
Service provider's signing certificate using SKI -->  
<ds:Signature>  
    ...  
    <ds:KeyInfo>  
        <wsse:SecurityTokenReference>  
            <wsse:KeyIdentifier  
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-  
wss-soap-message-security-1.0#Base64Binary"  
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-  
x509-token-profile-1.0#X509SubjectKeyIdentifier">  
                ...  
            </wsse:KeyIdentifier>  
        </wsse:SecurityTokenReference>  
    </ds:KeyInfo>  
</ds:Signature>  
    ...  
</wsse:Security>  
</soap:Header>  
<soap:Body>  
    <!-- This is encrypted data -->  
    ...  
</soap:Body>  
</soap:Envelope>
```

# 7 Metadata

## 7.1 WS-Addressing

### 7.1.1 Use of WS-Addressing 1.0

#### 7.1.1.1 Conformance criteria on service interface specifications

WS 7.1.1.1-1 Service interface specifications **MUST** use WS-Addressing 1.0 elements to carry addressing data in SOAP messages.

WS-Addressing 1.0 is defined by [WSA2006].

#### 7.1.1.2 Notes (non-normative)

WS-Addressing defines a mechanism for identifying messages and Web services endpoints. This information is used in the processing of the messages.

WS-Addressing is a W3C Recommendation, and is the standard mechanism for addressing in Web services. There are no alternative standards for Web services addressing.

WS-Addressing is being used in this profile as a mechanism for indicating the purpose of the SOAP message (see section 7.1.2 “WS-Addressing Action”) and for identifying SOAP messages (see section 7.1.3 “WS-Addressing MessageID”). WS-Addressing is used because the alternatives are not suitable. The alternative for WS-Addressing Action is to use SOAP Actions, but they are not recommended because of security reasons. The alternative for MessageID is to create a custom SOAP header that performs the same function as a standard WS-Security MessageID header—the use of standards is preferred over creating a non-standard mechanism.

Sections 7.1.4, 7.1.5 and 7.1.6 describe the use of WS-Addressing To, From, and ReplyTo headers. They are not needed by this profile. However, they are required by Web services toolkits, which is why they are included in the profile.

#### 7.1.1.3 Example (non-normative)

POLICY EXAMPLE

This example shows how to specify the use of WS-Addressing 1.0 in a WSDL.

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsm="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  ...
  <binding ...>
    <wsp:PolicyReference URI="#..." />
    ...
  </binding>
  ...
  <wsp:Policy wsu:Id="...">
    <wsam:Addressing/>
  </wsp:Policy>
</definitions>
```

## 7.1.2 WS-Addressing Action

### 7.1.2.1 Conformance criteria on service interface specifications

WS 7.1.2.1-1 Service interface specifications **MUST** define WS-Addressing Action values for all SOAP messages (i.e. input, output and fault messages) of all operations.

### 7.1.2.2 Conformance criteria on SOAP message creators

WS 7.1.2.2-1 SOAP message creators **MUST** include a WS-Addressing Action in every SOAP message.

### 7.1.2.3 Conformance criteria on service providers

WS 7.1.2.3-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criterion WS 7.1.2.2-1.

WS 7.1.2.3-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badWsaAction`) if the SOAP request violates criterion WS 7.1.2.2-1.

### 7.1.2.4 Notes (non-normative)

Web services need a mechanism for determining the purpose of a SOAP message.

This Action property is defined as a part of the WS-Addressing specification [WSA2006]. The use of WS-Addressing in WSDL is described in [WSAM2007].

The WS-Addressing binding for WSDL [WSAM2007] defines default values for WS-Addressing Action. However, these criteria recommend providing them explicitly in the WSDL to avoid any possible problems or confusion.

Section 7.1.3 recommends that SOAP messages provide a WS-Addressing Action value. Implementing these criteria requires that WSDL definitions specify WS-Addressing Action values for SOAP messages.

These criteria are the preferred mechanism for identifying operations.

### 7.1.2.5 Example (non-normative)

#### POLICY EXAMPLE

This example shows the inclusion of WS-Addressing Action for all input, output, and faults.

#### WSDL Definition:

```
<wsdl:definitions
  targetNamespace="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata">
  ...
  <wsdl:portType name="DSRPort">
    <wsdl:operation name="SendDS">
      <wsdl:input message="..."
wsam:Action="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0/DSRPort/SendDSR
equest"/>
      <wsdl:output message="..."

wsam:Action="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0/DSRPort/SendDSR
esponse"/>
      <wsdl:fault name="BadId" message="..."

wsam:Action="http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0:DSRPort/SendDS/
Fault:BadId"/>
    </wsdl:operation>
  ...
```

```
</wsdl:portType>
</wsdl:definitions>
```

### 7.1.3 WS-Addressing MessageID

#### 7.1.3.1 Conformance criteria on SOAP message creators

- WS 7.1.3.1-1 SOAP message creators **MUST** include a WS-Addressing MessageID in all SOAP messages (SOAP requests, responses, and faults).
- WS 7.1.3.1-2 SOAP message creators **MUST** use a value that is globally unique for the WS-Addressing MessageID header.
- WS 7.1.3.1-3 SOAP message creators **MUST NOT** send more than one SOAP message containing the same WS-Addressing MessageID value.
- WS 7.1.3.1-4 SOAP message creators **SHOULD** use a UUID formatted as a URN, as specified by [RFC4122], as the value of the MessageID header.

#### 7.1.3.2 Conformance criteria on SOAP message consumers

- WS 7.1.3.2-1 SOAP message consumers **MUST** reject SOAP messages that do not contain a WS-Addressing MessageID.
- WS 7.1.3.2-2 SOAP message consumers **SHOULD** reject SOAP messages with duplicate WS-Addressing MessageID values.

#### 7.1.3.3 Conformance criteria on service providers

- WS 7.1.3.3-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criteria WS 7.1.3.1-1, WS 7.1.3.1-2, WS 7.1.3.1-3, or WS 7.1.3.1-4.
- WS 7.1.3.3-2 Service providers **SHOULD** respond with a standardError SOAP fault (with an error code of badWsaMessageId) if the SOAP request violates criteria WS 7.1.3.1-1, WS 7.1.3.1-2, WS 7.1.3.1-3, or WS 7.1.3.1-4.

#### 7.1.3.4 Notes (non-normative)

The WS-Addressing MessageID header is to be used for identifying related messages. Namely, for associating the SOAP response with the corresponding SOAP request.

The WS-Addressing MessageID header can also be used for duplicate message detection. This is duplicate detection at the SOAP message level. This is different from any application level duplicate detection that might also be needed.

Duplicate WS-Addressing MessageID headers could occur because of malicious replay attacks on the system. It could also occur because of an implementation error.

The inclusion of the WS-Addressing MessageID header is mandatory for SOAP message creators, because they do not know if the SOAP message consumer needs to use it or not.

The WS-Addressing MessageID value must be unique for every SOAP message. This includes SOAP messages which are resent. The resent SOAP message is treated as a different SOAP message with its own unique MessageID value.

## UUID URNs

UUIDs have been suggested here as the preferred approach for obtaining identifiers which can satisfy the globally unique requirements for the MessageID value.

The Universally Unique Identifier (UUID) expressed as a URN is described by [RFC4122]. The syntax for a URN is defined in [RFC2141], and consists of the following:

- A leading sequence "urn:", followed by
- A Namespace Identifier, followed by
- A colon character, followed by
- A Namespace Specific String.

In [RFC4122] the Namespace Identifier is defined as "UUID" and the syntax for the Namespace Specific String is defined to be outputted in lower-case letters.

Be aware that URN UUIDs need to be processed in a case-insensitive manner. The [RFC2141] states that the leading "urn:" and the Namespace Identifier are both case-insensitive. The [RFC4122] states that the Namespace Specific String is case-insensitive on input. Therefore, the following are examples of equivalent URNs:

```
urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
urn:UUID:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
URN:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
URN:UUID:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
Urn:UUId:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
```

The following is an invalid UUID URN, but needs to be processed as if it was a correct UUID URN by treating the uppercase letters as lowercase:

```
urn:uuid:F81d4FAE-7DEC-11D0-A765-00a0c91e6bf6 (invalid example)
```

## UUID URI

Another method of representing a UUID is as a URI with a UUID URI scheme. The syntax consists of the following:

- A leading sequence "uuid", followed by
- The UUID.

For example:

```
uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6 (not recommended)
```

This format is based on an out-of-date proposal, but is implemented by some toolkits. Although permitted by the criteria, it is recommended that a UUID URN be used if possible, since it is an official IETF RFC.

## 7.1.4 WS-Addressing in SOAP requests

### 7.1.4.1 Conformance criteria on service invokers

WS 7.1.4.1-1 Service invokers **MUST** include the WS-Addressing To header in SOAP request messages.

### 7.1.4.2 Conformance criteria on service providers

WS 7.1.4.2-1 Service providers **SHOULD** respond with a SOAP fault if the SOAP request violates criteria WS 7.1.4.1-1.

- WS 7.1.4.2-2 Service providers **SHOULD** respond with a `standardError` SOAP fault (with an error code of `badWsaTo`) if the SOAP request violates criterion WS 7.1.4.1-1.
- WS 7.1.4.2-3 Service providers **MUST NOT** expect the WS-Addressing `To` header to contain any particular value.

#### 7.1.4.3 Notes (non-normative)

When combined with criteria in section 7.1.2 and 7.1.3, the mandatory WS-Addressing headers for a SOAP request are: `To`, `Action`, and `MessageID`.

##### WS-ADDRESSING `To` HEADER

The WS-Addressing `To` header is required by some Web service toolkits when the WS-Addressing functionality is switched on. To interoperate with these toolkits, the `To` header must be provided.

Many toolkits put the URL of the service being invoked in the WS-Addressing `To` header. If the SOAP message has been routed through a different machine then the WS-Addressing `To` header will not match the physical URL of the service provider. For example, this might happen if the service invoker interacts with a proxy in the DMZ which then forwards the SOAP request to a service instance running on a different machine inside the organisation. Also, the service invoker is permitted by the WS-Addressing specification to put the value of `"http://www.w3.org/2005/08/addressing/anonymous"` in the WS-Addressing `To` header.

Therefore, service providers needs to ignore the value in the WS-Addressing `To` header. They cannot always expect the value to match their deployed URL.

##### OTHER WS-ADDRESSING HEADERS

The other WS-Addressing headers (`From`, `ReplyTo` and `FaultTo`) are not required and how the toolkits process them is not well defined. It is suggested that they are not used, unless their behaviour is defined in the service interface specification.

The WS-Addressing `Action` header can be used to route the request messages. For example, toolkits can map SOAP messages to operation calls using the `Action` value. Intermediaries can specify particular routes for messages based on their `Action` value. The `Action` values should follow the rules in section 7.1.2.

The WS-Addressing `MessageID` header is included to support auditing of SOAP messages.

##### DIGITAL SIGNING

In keeping with section 6.2.3, all WS-Addressing headers must be digitally signed.

#### 7.1.4.4 Example (non-normative)

##### SOAP MESSAGE EXAMPLE

This example shows the required WS-Addressing headers in a SOAP request.

##### SOAP Request:

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soap:Header>
    ...
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
  </soap:Header>
  <wsa:Action>http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0/DSRPort/SendDSRRequest</wsa:Action>
```

```

<wsa:MessageID>urn:uuid:652d329a-cd1e-11db-8314-0800200c9a66</wsa:MessageID>
...
</soap:Header>
<soap:Body> ... </soap:Body>
</soap:Envelope>

```

## 7.1.5 WS-Addressing in SOAP responses

### 7.1.5.1 Conformance criteria on service providers

- WS 7.1.5.1-1 Service providers **MUST** include a WS-Addressing `RelatesTo` header in SOAP response messages to indicate the SOAP request message that it is a reply for.
- WS 7.1.5.1-2 Service providers **MAY** have additional WS-Addressing `RelatesTo` header in a SOAP response message.

### 7.1.5.2 Notes (non-normative)

When combined with criteria in section 7.1.2 and 7.1.3, the mandatory WS-Addressing headers for a SOAP response are: `RelatesTo`, `Action`, and `MessageID`.

The value of the `MessageID` header in the SOAP response message must be a new globally unique value generated by the service provider. It is not the same value that was used in the corresponding SOAP request message.

#### DIGITAL SIGNING

In keeping with 6.2.3, all of these WS-Addressing headers must be included in the data that is digitally signed.

### 7.1.5.3 Example (non-normative)

#### SOAP MESSAGE EXAMPLE

This example shows the required WS-Addressing headers in a SOAP response.

#### SOAP Response:

```

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soap:Header>
  ...

  <wsa:Action>http://ns.nehta.gov.au/Example/Svc/ProfileExample/1.0/DSRPort/SendDSR
  sponse</wsa:Action>
    <wsa:MessageID>urn:uuid:037dfbd0-cd1e-11db-8314-0800200c9a66</wsa:MessageID>
    <wsa:RelatesTo
  RelationshipType="http://www.w3.org/2005/08/addressing/reply">urn:uuid:652d329a-
  cd1e-11db-8314-0800200c9a66</wsa:RelatesTo>
  ...
  </soap:Header>
  <soap:Body> ... </soap:Body>
</soap:Envelope>

```

## 7.1.6 WS-Addressing in SOAP faults

### 7.1.6.1 Conformance criteria on service providers

- WS 7.1.6.1-1 Service providers **MUST** include a WS-Addressing `RelatesTo` header in SOAP fault messages to indicate the SOAP request message that it is a fault for.

WS 7.1.6.1-2 Service providers **MAY** have additional WS-Addressing `RelatesTo` headers in a SOAP fault message.

#### 7.1.6.2 Notes (non-normative)

When combined with criteria in section 7.1.2 and 7.1.3, the mandatory WS-Addressing headers for a SOAP faults are: `RelatesTo`, `Action`, and `MessageID`.

The value of the `MessageID` header in the SOAP response message must be a new globally unique value generated by the service provider. It is not the same value that was used in the corresponding SOAP request message.

#### DIGITAL SIGNING

In keeping with 6.2.3, all of these WS-Addressing headers must be included in the data that is digitally signed.

# 8 TLS security

## 8.1 PKI for TLS

### 8.1.1 RSA certificates

#### 8.1.1.1 Obligations on service invokers

WS 8.1.1.1-1 Service invokers **MUST** use X.509v3 RSA certificates.

#### 8.1.1.2 Obligations on service providers

WS 8.1.1.2-1 Service providers **MUST** use X.509v3 RSA certificates.

#### 8.1.1.3 Notes (non-normative)

The Public Key Infrastructure (PKI) is a standard for key management for asymmetric cryptographic algorithms.

Asymmetric cryptographic algorithms use two keys which are mathematically related. These keys are referred to as “key pairs”. This is different from symmetric cryptographic algorithms which use only one key.

One of the two keys is designated the “public key” and the other the “private key”. The private key is kept by the key owner and not disclosed to any other party. The public key can be widely disclosed to other parties.

The standard for representing the public key is X.509v3.

RSA is an asymmetric cryptographic algorithm that is commonly used for the public-private key pairs issued for PKI certificates.

### 8.1.2 Key usage

#### 8.1.2.1 Conformance criteria on service invokers

WS 8.1.2.1-1 Service invokers **MUST** use X.509v3 certificates that either:

- Do not contain a *key usage* certificate extension, or
- Do contain a *key usage* certificate extension that **MUST** at least have the *digitalSignature* bit set.

#### 8.1.2.2 Conformance criteria on service providers

WS 8.1.2.2-1 Service providers **MUST** use X.509v3 certificates that either:

- Do not contain a key usage certificate extension, or
- Do contain a key usage certificate extension that **MUST** at least have the *keyEncipherment* bit set.

#### 8.1.2.3 Notes (non-normative)

The client certificate must be used for digital signing, as required by the TLS specification (section 7.4.8 of [RFC2246]) for RSA.

The server certificate needs to be used for encryption by having the *keyEncipherment* bit set. This is a requirement for using the RSA key exchange algorithm (section 7.4.2 of [RFC2246]) and to avoid the use of temporary RSA keys (section D.1 of [RFC2246]).

A description of key usage can be found in section 4.2.1.3 of [RFC3280] or in section 8.2.2.3 of [X509-2005].

### 8.1.3 Certificates with identifiers

#### 8.1.3.1 Conformance criteria on service invokers

WS 8.1.3.1-1 Service invokers **MUST** use X.509v3 certificates that can be associated with the identity of the service invoker.

#### 8.1.3.2 Conformance criteria on service providers

WS 8.1.3.2-1 Service providers **MUST** use X.509v3 certificates that can be associated with the identity of the service provider.

#### 8.1.3.3 Notes (non-normative)

A unique identifier for the entity is necessary to ensure the correct relationship between the message, the service sending or receiving the message, and the organisation operating the service. If this relationship cannot be established, there is a high security risk for fraud and impersonation attacks.

These criteria ensure that there is always a mechanism to uniquely associate a certificate to its owner, that this mechanism will be present, and that all parties know how and when to use it.

## 8.2 Transport Layer Security

### 8.2.1 Protocol

#### 8.2.1.1 Obligations on service invokers

WS 8.2.1.1-1 Service invokers **MUST** support a connection using Transport Layer Security (TLS) 1.0 protocol.

TLS 1.0 is defined by [RFC2246].

#### 8.2.1.2 Obligations on service providers

WS 8.2.1.2-1 Service providers **MUST** be able to establish a connection using Transport Layer Security (TLS) 1.0.

WS 8.2.1.2-2 Service providers **MUST NOT** establish connections using Secure Sockets Layer (SSL) 2.0.

#### 8.2.1.3 Notes (non-normative)

Transport Layer Security (TLS) is a protocol for establishing a secured channel for communications. It is used to encrypt the data for confidentiality. It is also used to authenticate one or both parties.

This section will refer to both SSL and TLS as different versions of the same protocol, with SSL considered as an older version of the TLS protocol.

The first publicly released version of the protocol was SSL 2.0. However, SSL 2.0 has known security flaws and must not be used. It is explicitly prohibited, to ensure that service providers cannot be tricked into using it.

SSL 3.0 [SSL30] addresses those security flaws and was later formalised as TLS 1.0. These criteria have chosen TLS 1.0 as the baseline version that all service invokers and service providers must support.

There are newer versions of TLS: namely TLS 1.1 [RF4346] and TLS 1.2 [RFC5246]. These can be used if both entities support them. However, they are not mandated by these criteria because they are currently not widely

implemented. These newer versions of the protocol can be used, if both entities support them.

The TLS protocols negotiate the protocol version (as well as the cipher suites) used. The negotiation consists of the client indicating what it can support to the server, in a `ClientHello` message. The server then responds by picking one of them to use.

These criteria are concerned with the protocol that is picked by the server to use, and do not constrain how the negotiation is performed. It is possible for a client to use a SSL 2.0 `ClientHello` message, as long as that `ClientHello` message indicates that it can also support TLS 1.0. A server can accept SSL 2.0 `ClientHello` messages, as long as it does not pick SSL 2.0 as the negotiated protocol to use.

## 8.2.2 Mutual authentication

### 8.2.2.1 Obligations on service invokers

WS 8.2.2.1-1 Service invokers **MUST** support a mutually authenticated connection.

### 8.2.2.2 Obligations on service providers

WS 8.2.2.2-1 Service providers **MUST** use a mutually authenticated connection.

WS 8.2.2.2-2 Service providers **MUST NOT** accept connections which are not mutually authenticated.

### 8.2.2.3 Notes (non-normative)

Mutual authentication allows the service provider (i.e. server) to authenticate the service invoker (i.e. client). With all TLS connections, the service invoker (i.e. client) also authenticates the service provider (i.e. server).

Mutual authentication is used so that both parties can verify who they are communicating with.

## 8.2.3 Cipher suites

### 8.2.3.1 Obligations on service providers

WS 8.2.3.1-1 Service providers **MUST NOT** select a cipher suite with no data encryption.

WS 8.2.3.1-2 Service providers **MUST NOT** select a cipher suite that uses the anonymous Diffie-Hellman algorithm.

WS 8.2.3.1-3 Service providers **MUST NOT** select a cipher suite with a reduced key length to comply with previous US export restrictions.

WS 8.2.3.1-4 Service providers **MUST NOT** select a cipher suite that uses single DES for data encryption.

### 8.2.3.2 Notes (non-normative)

These criteria are designed to prevent insecure cipher suites from being used.

The algorithm suites listed below are only examples from the TLS 1.0 specification. The SSL 3.0 specification contains the same algorithm suite (but their names start with "SSL" instead of "TLS"). This is not a complete list of all algorithm suites to avoid. There are other algorithm suites defined outside of TLS 1.0 that are also prohibited by these criteria.

Some cipher suites do not perform encryption (they are only used for integrity and/or authentication). Obviously these do not provide any confidentiality and are prohibited by these criteria. For example:

- TLS\_RSA\_WITH\_NULL\_MD5 { 0x00, 0x01 }
- TLS\_RSA\_WITH\_NULL\_SHA { 0x00, 0x02 }

Some cipher suites use the anonymous Diffie-Hellman algorithm, which does not provide authentication of the parties involved. Since this makes the protocol vulnerable to man-in-the-middle attacks they are prohibited by these criteria. For example:

- TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5 { 0x00, 0x17 }
- TLS\_DH\_anon\_WITH\_RC4\_128\_MD5 { 0x00, 0x18 }
- TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x19 }
- TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA { 0x00, 0x1A }
- TLS\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA { 0x00, 0x1B }

Some cipher suites were weakened to satisfy US export restrictions. When SSL 3.0 and TLS 1.0 were created, the US had restrictions on the export of software with strong cryptographic algorithms. The standards defined a number of weaker cipher suites with reduced key lengths to comply with those export restrictions. Since then, the US export restrictions have weakened while processing power have increased, making these cipher suites unsuitable for securing communications. These are the ones with "EXPORT" in their names, for example:

- TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 { 0x00, 0x03 }
- TLS\_RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5 { 0x00, 0x06 }
- TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x08 }
- TLS\_DH\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x0B }
- TLS\_DH\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x0E }
- TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x11 }
- TLS\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x14 }
- TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5 { 0x00, 0x17 }
- TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA { 0x00, 0x19 }

Note: TLS 1.1 also prohibits the use of the above export algorithms.

Some cipher suites use single DES which has a 56-bit key size. This algorithm is no longer considered secure, because it can be broken using brute force techniques. Examples of cipher suites using DES are:

- TLS\_RSA\_WITH\_DES\_CBC\_SHA { 0x00, 0x09 }
- TLS\_DH\_DSS\_WITH\_DES\_CBC\_SHA { 0x00, 0x0C }
- TLS\_DH\_RSA\_WITH\_DES\_CBC\_SHA { 0x00, 0x0F }
- TLS\_DHE\_DSS\_WITH\_DES\_CBC\_SHA { 0x00, 0x12 }
- TLS\_DHE\_RSA\_WITH\_DES\_CBC\_SHA { 0x00, 0x15 }
- TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA { 0x00, 0x1A }

Triple DES is still considered strong and can be used. Triple DES is an algorithm that applies the DES algorithm three times, effectively increasing the key length and making the algorithm less susceptible to brute force attacks.

# Appendix A: Informative references

- [ACSI33] Defence Signals Directorate, *Australian Government Information and Communications Technology Security Manual*, ACSI 33, 2005-09-19.
- [BP2006] WS-I, *Basic Profile 1.1*, Final Material, 10 April 2006, <http://www.ws-i.org/Profiles/BasicProfile-1.1-2006-04-10.html>.
- [BP2007] WS-I, *Basic Profile 2.0*, Working Group Draft, 25 October 2007, [http://www.ws-i.org/Profiles/BasicProfile-2\\_0\(WGD\).html](http://www.ws-i.org/Profiles/BasicProfile-2_0(WGD).html).
- [BSP2007] WS-I, *Basic Security Profile 1.0*, Final Material, 30 March 2007, <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2007-03-30.html>.
- [BUTE2005] Russell Butek, *Which Style of WSDL Should I Use?*, IBM, 24 May 2005, <http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/#N1021F>
- [DAVI2001a] Donald T. Davis, *Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML*, Proc. Usenix Tech. Conf. 2001 (Boston, Mass., 25-30 June 2001), pp. 65-78, [http://world.std.com/~dtd/sign\\_encrypt/sign\\_encrypt7.PDF](http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.PDF)
- [DAVI2001b] Don Davis, *Defective Sign-and-Encrypt: Can you really trust S/MIME, PKCS#7, PGP, and XML*, Dr. Dobb's Journal #330, v.26 (11), November 2001, pp. 30-36, <http://www.ddj.com/security/184404841>
- [DSIG2002] W3C, *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002, <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>
- [GIIWS2007] NEHTA, *Guidelines for Implementing Interoperable Web Services*, version 1.0, 2007-03-28.
- [INFO2004] W3C, *XML Information Set (Second Edition)*, W3C Recommendation, 4 February 2004, <http://www.w3.org/TR/2004/REC-xml-infoset-20040204>
- [RFC3280] IETF, *RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, R. Housley, W. Polk, W. Ford and D. Solo, April 2002, <http://ietf.org/rfc/rfc3280.txt>
- [RFC3986] IETF, *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, January 2005, <http://ietf.org/rfc/rfc3986.txt>
- [SOAP2000] W3C, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, 8 May 2000, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- [SOAP2003a] W3C, *SOAP Version 1.2 Part 0: Primer (Second Edition)*, W3C Recommendation, 27 April 2007, <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>
- [XENC2002] W3C, *XML Encryption Syntax and Processing*, W3C Recommendation, 10 December 2002, <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>

# Appendix B: NEHTA SOAP faults

This appendix defines the NEHTA standard SOAP faults.

This NEHTA set of SOAP faults have been defined to improve compatibility when handling error conditions arising from the criteria defined in this document. It gives guidance to Web service providers for what to do in the event of an error. It also gives Web service consumers guidance on how to detect when a particular error event has occurred.

## B.1 SOAP faults

All the faults defined in this document have the following attributes:

- `errorCode`

a string indicating the error code of the fault. This is mandatory. See below for the set of permitted enumerated values.
- `message`

human readable text which describes the problem. This is mandatory.

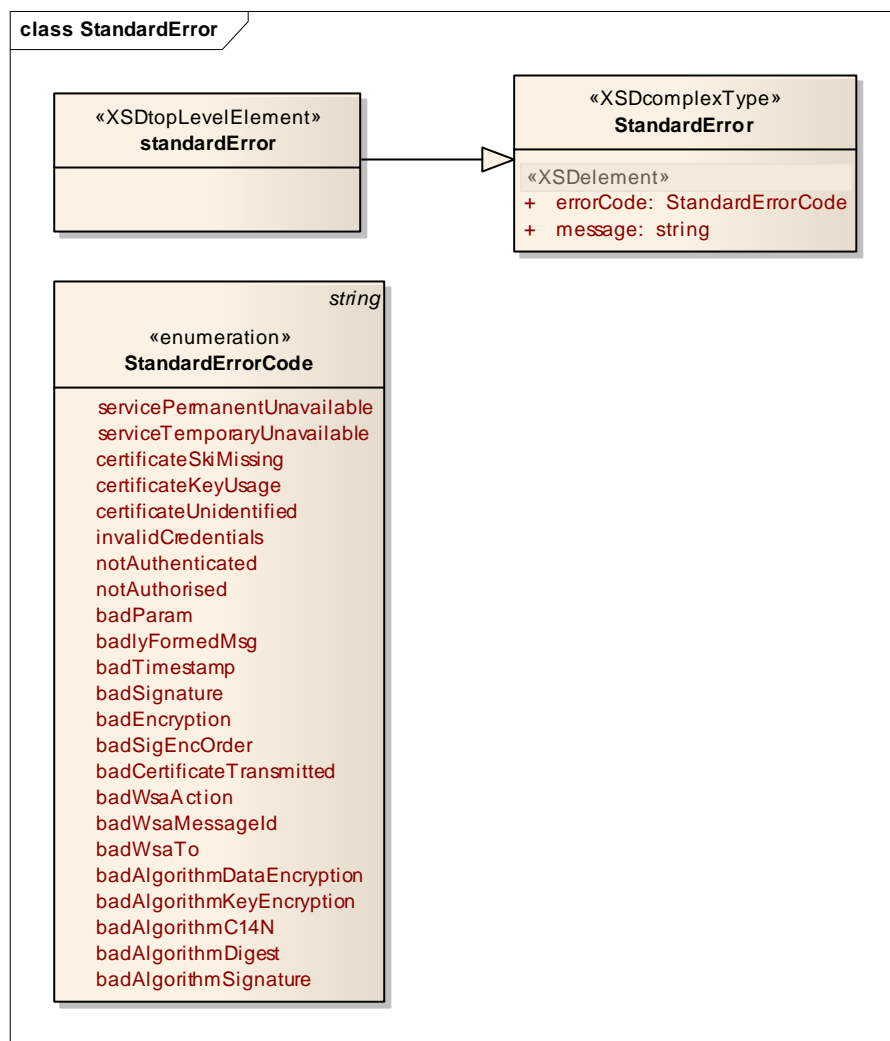


Figure 3: Class diagram of error data structures

## B.2 Standard error codes

The NEHTA standard errors can be classified into these categories of causes:

- Service errors
- Certificate type errors
- Security errors
- Bad request errors

### B.2.1 Service errors

Service errors are used to indicate a problem with the service provider. They are not caused by the service invoker doing anything wrong and there is no way for the service invoker to resolve the problem.

#### B.2.1.1 `servicePermanentUnavailable`

This error indicates that a permanent problem has occurred in the service provider program.

This error indicates that the operation did not succeed. The operation will not succeed if it is retried.

#### B.2.1.2 `serviceTemporaryUnavailable`

This error indicates that a temporary problem has occurred in the service provider program.

This error indicates that the operation did not succeed. The operation might succeed if it is retried at a different time.

### B.2.2 Certificate type errors

Certificate type errors indicate that the wrong kind of PKI certificate was used. The user should pick a different key/certificate to use.

#### B.2.2.1 `certificateSkiMissing`

A certificate used does not have a Subject Key Identifier.

#### B.2.2.2 `certificateKeyUsage`

A certificate used has an unacceptable Key Usage extension.

#### B.2.2.3 `certificateUnidentified`

A certificate used does not identify the party it is associated with.

### B.2.3 Security errors

Security errors indicate some form of configuration or operational error relating to security.

#### B.2.3.1 `invalidCredentials`

The credentials cannot be used because they are not valid. This includes expired and revoked credentials.

#### B.2.3.2 `notAuthenticated`

The service invoker can be identified but not successfully authenticated.

### B.2.3.3 notAuthorised

The service invoker has been successfully authenticated, but is not authorised to invoke the Web service operation with the given set of parameters.

## B.2.4 Bad request errors

Bad request errors indicate that the service invoker has sent an incorrect SOAP request. They indicate a problem that only the author of the service invoker software can resolve.

Although it is desirable from an interoperability point-of-view to have standard error codes for bad requests, most Web service toolkits return proprietary SOAP faults for several of these errors. The programmer may have little or no control over the SOAP faults that are returned for these errors.

### B.2.4.1 badParam

The parameters provided to the service are incorrect.

### B.2.4.2 badlyFormedMsg

The SOAP request did not contain the expected message format.

This error indicates that syntax of the SOAP request was not correct. This means it did not match the service specification. For example, the XML in the SOAP request was not well formed or it was not valid according to the XML Schema definition.

### B.2.4.3 badTimestamp

The WS-Security `Created` timestamp is missing or has the wrong format.

### B.2.4.4 badSignature

The SOAP request has not been signed, or is signed incorrectly.

### B.2.4.5 badEncryption

The SOAP request has not been encrypted, or is encrypted incorrectly.

### B.2.4.6 badSigEncOrder

The SOAP request was encrypted before it was signed (when it should have been signed and then encrypted).

### B.2.4.7 badCertificateTransmit

The certificates for signing and/or encrypting are not correctly referenced in the SOAP request.

### B.2.4.8 badWsaAction

The WS-Addressing `Action` element is missing or has an incorrect value.

### B.2.4.9 badWsaMessageId

The WS-Addressing `MessageID` element is missing or has an incorrect value.

### B.2.4.10 badWsaTo

The WS-Addressing `To` element is missing.

**B.2.4.11 badAlgorithmDataEncryption**

The symmetric algorithm used for encrypting the data is not acceptable.

**B.2.4.12 badAlgorithmKeyEncryption**

The algorithm used for encrypting the symmetric key is not acceptable.

**B.2.4.13 badAlgorithmC14N**

The algorithm used for canonicalizing the data is not acceptable.

**B.2.4.14 badAlgorithmDigest**

The algorithm used for calculating the digest is not acceptable.

**B.2.4.15 badAlgorithmSignature**

The algorithm used for signing is not acceptable.

**B.3 XML Schema**

The following XML Schema defines the types and elements used by the NEHTA SOAP faults.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://ns.nehta.gov.au/CoreConnectivity/Xsd/StandardError:1.0"
  targetNamespace="http://ns.nehta.gov.au/CoreConnectivity/Xsd/StandardError:1.0"
  elementFormDefault="qualified">
  <xsd:simpleType name="StandardErrorCode">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="servicePermanentUnavailable"/>
      <xsd:enumeration value="serviceTemporaryUnavailable"/>
      <xsd:enumeration value="certificateSkiMissing"/>
      <xsd:enumeration value="certificateKeyUsage"/>
      <xsd:enumeration value="certificateUnidentified"/>
      <xsd:enumeration value="invalidCredentials"/>
      <xsd:enumeration value="notAuthenticated"/>
      <xsd:enumeration value="notAuthorised"/>
      <xsd:enumeration value="badParam"/>
      <xsd:enumeration value="badlyFormedMsg"/>
      <xsd:enumeration value="badTimestamp"/>
      <xsd:enumeration value="badSignature"/>
      <xsd:enumeration value="badEncryption"/>
      <xsd:enumeration value="badSigEncOrder"/>
      <xsd:enumeration value="badCertificateTransmitted"/>
      <xsd:enumeration value="badWsaAction"/>
      <xsd:enumeration value="badWsaMessageId"/>
      <xsd:enumeration value="badWsaTo"/>
      <xsd:enumeration value="badAlgorithmDataEncryption"/>
      <xsd:enumeration value="badAlgorithmKeyEncryption"/>
      <xsd:enumeration value="badAlgorithmC14N"/>
      <xsd:enumeration value="badAlgorithmDigest"/>
      <xsd:enumeration value="badAlgorithmSignature"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="StandardError">
    <xsd:sequence>
      <xsd:element name="errorCode" type="tns:StandardErrorCode"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="message" type="xsd:string"
        minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="standardError" type="tns:StandardError"/>
</xsd:schema>
```

# Appendix C: Change log

## Version 3.1

- Added a discussion about why the set of standard SOAP faults were defined.
- Clarified WS 6.2.3.2-10 and WS 6.2.4.4-3.
- Added WS 6.2.7.3-1 to place obligation on service interface specification to define the rules for validating certificates.
- Added TLS security profile.
- Added WS 6.2.3.3-3 and WS 6.2.4.4-3 to clarify which key and certificate are used for signing and encrypting SOAP responses.

## Version 3.0

- Introduced formal criteria for the standards that were previously recommended in the NEHTA *Web Services Standards Profile 2.0* to enable compliance and conformance checking.
- Introduced formal criteria for the guidelines that were previously listed in the NEHTA *Guidelines for Implementing Interoperable Web Services 1.0* to enable compliance and conformance checking.
- Removed the SOAP Message Transmission Optimization Mechanism (MTOM) standard.
- Removed the XML-binary Optimized Packaging (XOP) standard.
- Changed criteria for structure of WSDL files into two separate files.
- Added use of WS-Policy 1.5.
- Added use of WS-Addressing 1.0 Metadata.
- Added use of WS-SecurityPolicy 1.2.
- Added criteria for URN in namespace values.
- Added criteria for using persistent HTTP connections.
- Changed criteria for SOAP Action usage.
- Added criteria on SOAP fault behaviour and standard faults.
- Added criteria for PKI and the types of certificates used.
- Changed criteria for WS-Security timestamps.
- Changed criteria for WS-Security algorithm suite to `Basic256Rsa15`.
- Changed criteria for WS-Addressing `To` and `From` elements.
- Added standard error conditions and SOAP faults.