

Web Services Secure Conversation Language (WS-SecureConversation)

Version 1.0

December 18, 2002

Authors

Giovanni Della-Libera, Microsoft
Brendan Dixon, Microsoft
Praerit Garg, Microsoft
Satoshi Hada, IBM
Phillip Hallam-Baker, VeriSign
Maryann Hondo, IBM
Chris Kaler (Editor), Microsoft
Hiroshi Maruyama, IBM
Anthony Nadalin (Editor), IBM
Nataraj Nagaratnam, IBM
Andrew Nash, RSA Security
Rob Philpott, RSA Security
Hemma Prafullchandra, VeriSign
John Shewchuk, Microsoft
Dan Simon, Microsoft
Elliot Waingold, Microsoft
Riaz Zolfonoon, RSA Security

Copyright Notice

(c) 2001, 2002 International Business Machines Corporation, Microsoft Corporation, RSA Security Inc, VeriSign Inc. All rights reserved.

IBM, Microsoft, RSA, and VeriSign (collectively, the "Authors") hereby grant you permission to copy and display the WS-SecureConversation Specification, in any medium without fee or royalty, provided that you include the following on ALL copies of the WS-SecureConversation Specification, or portions thereof, that you make:

1. A link or URL to the Specification at this location
2. The copyright notice as shown in the WS-SecureConversation Specification.

EXCEPT FOR THE COPYRIGHT LICENSE GRANTED ABOVE, THE AUTHORS DO NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY, INCLUDING PATENTS, THEY OWN OR CONTROL.

THE WS-SecureConversation SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE WS-SecureConversation SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE WS-SecureConversation SPECIFICATION.

The WS-SecureConversation Specification may change before final release and you are cautioned against relying on the content of this specification.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without specific, written prior permission. Title to copyright in the WS-SecureConversation Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.

Abstract

This specification defines extensions that build on [WS-Security](#) to provide secure communication. Specifically, we define mechanisms for establishing and sharing security contexts, and deriving session keys from security contexts.

Modular Architecture

By using the XML, SOAP and WSDL extensibility models, the WS* specifications are designed to be composed with each other to provide a rich Web services environment. WS-SecureConversation by itself does not provide a complete security solution for Web services. WS-SecureConversation is a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models.

Status

This WS-SecureConversation Specification is an initial public draft release and is provided for review and evaluation only. IBM, Microsoft, RSA, and VeriSign hope to solicit your contributions and suggestions in the near future. IBM, Microsoft, RSA, and VeriSign make no warranties or representations regarding the specifications in any manner whatsoever.

Table of Contents

1. Overview
 - 1.1. Goals and Non-Goals
 - 1.2. Requirements
2. Notations and Terminology
 - 2.1 Notational Conventions
 - 2.2 Namespace
 - 2.3. Schema and WSDL Files
 - 2.4. Terminology
3. Security Context Token
4. Establishing Security Context
5. Deriving Keys
6. Error Handling
7. Security Considerations

- 8. Acknowledgements
- 9. References

1. Overview

The mechanisms defined in [WS-Security](#) provide the basic mechanisms on top of which secure messaging can be defined. This specification defines extensions to allow security context establishment and sharing, session key derivation.

The [WS-Security](#) specification focuses on the message authentication model. This approach, while useful in many situations, is subject to several forms of attack. Accordingly, this specification introduces a security context and its usage. The context authentication model authenticates a series of messages thereby addressing these shortcomings, but requires additional communications if authentication happens prior to normal application exchanges.

To implement these models (described below) we introduce new headers and SOAP extensions.

Compliant services are NOT REQUIRED to implement everything defined in this specification. However, if a service implements an aspect of the specification, it MUST comply with the requirements specified (e.g. related "MUST" statements).

1.1. Goals and Non-Goals

The primary goals of this specification are:

- Define how security contexts are established
- Specify how derived keys are computed and passed

It is not a goal of this specification to define how trust is established or determined.

1.2. Requirements

The following list identifies the key driving requirements:

- Derived keys and per-message keys
- Extensible security contexts

2. Notations and Terminology

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

2.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119](#).

Namespace URIs of the general form "some-URI" represents some application-dependent or context-dependent URI as defined in [RFC2396](#).

2.2 Namespace

The [XML namespace](#) URI that MUST be used by implementations of this specification is:

<http://schemas.xmlsoap.org/ws/2002/12/secext>

The following namespaces are used in this document:

Prefix	Namespace
S	http://schemas.xmlsoap.org/soap/envelope/
wsu	http://schemas.xmlsoap.org/ws/2002/07/utility
wsse	http://schemas.xmlsoap.org/ws/2002/12/secext
wsp	http://schemas.xmlsoap.org/ws/2002/12/policy
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#

2.3. Schema File

The schema for this specification can be located at:

<http://schemas.xmlsoap.org/ws/2002/12/secext>

In this document reference is made to the `wsu:Id`, `wsu:Created` and `wsu:Expires` attributes in a utility schema (<http://schemas.xmlsoap.org/ws/2002/07/utility>). The `wsu:Id` attribute, `wsu:Created` and `wsu:Expires` attributes were added to the utility schema with the intent that other specifications requiring such an ID or timestamp could reference it (as is done here).

2.4. Terminology

We provide basic definitions for the security terminology used in this specification. Note that readers should be familiar with the [WS-Security](#) specification.

Claim – A *claim* is a statement that a client signer makes (e.g. name, identity, key, group, privilege, capability, etc) about the client.

Security Token – A *security token* represents a collection of claims.

Signed Security Token – A *signed security token* is a security token that is asserted and cryptographically endorsed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket).

Proof-of-Possession Token – A *proof-of-possession token* is a security token that contains data that a sending party can use to demonstrate proof-of-possession of another security token or a shared secret. Typically, although not exclusively, the proof-of-possession information is encrypted with a key known only to the sender and recipient parties.

Digest – A *digest* is a cryptographic checksum of an octet stream.

Signature - A *signature* is a cryptographic binding of a key and a digest of some information. This covers both symmetric key-based and public key-based signatures. Examples of information that can be signed include messages and security tokens.

Security Token Service - A *security token service* is a Web service that issues security tokens (see [WS-Security](#)). That is, it makes assertions based on evidence that it trusts, to whoever trusts it. To communicate trust, a service requires proof, such as a security token or set of security tokens, and issues a security token with its own trust statement (note that for some security token formats this can just be a re-issuance or co-signature). This forms the basis of trust brokering.

3. Security Context Token

While message authentication is useful for simple or one-way messages, parties that wish to exchange multiple messages typically establish a secure security context in which to exchange multiple messages. A security context is shared among the communicating parties for the lifetime of a communications association.

In this specification, a security context is represented by the `<SecurityContextToken>` security token.

Once the context and secret have been established (authenticated), the mechanisms described in [Derived Keys](#) can be used to compute derived keys for each key usage in the secure context.

The following represents an overview of the syntax of the `<SecurityContextToken>` element. It should be noted that this token supports an open content model to allow context-specific data to be passed.

```
<wsse:SecurityContextToken wsu:Id="...">
  <wsu:Identifier>...</wsu:Identifier>
  <wsu:Created>...</wsu:Created>
  <wsu:Expires>...</wsu:Expires>
  <wsse:Keys>
    <xenc:EncryptedKey Id="...">...</xenc:EncryptedKey>
    <wsse:SecurityTokenReference>...</wsse:SecurityTokenReference>
    ...
  </wsse:Keys>
</wsse:SecurityContextToken>
```

The following describes elements and attributes used in a `<wsse:SecurityContextToken>` element.

`/SecurityContextToken`

This element is a security token that describes a security context.

`/SecurityContextToken/wsu:Identifier`

This required element identifies the security context using a URI. Each security context URI **MUST** be globally unique to both the sender and recipient.

`/SecurityContextToken/wsu:Created`

This optional element indicates the creation time of the security context. This is typically only specified on the first usage of the token. That is, it is typically cached as part of the context by the recipient.

/SecurityContextToken/wsu:Expires

This optional element indicates the expiration time of the security context according to the requestor's clock. This is typically only specified on the first usage of the token. That is, it is typically cached as part of the context by the recipient.

/SecurityContextToken/Keys

This optional element holds the shared secrets of the security context. This is typically only specified on the first usage of the token. That is, it is typically cached as part of the context by the recipient. If there is no <Keys> element, then the shared secret is assumed to be already known and associated with the security context identified by the URI specified in the <Identifier> element.

/SecurityContextToken/Keys/xenc:EncryptedKey

This optional element holds the shared secret of the security context.

/SecurityContextToken/Keys/xenc:EncryptedKey/@Id

This optional attribute specifies an "ID" for the key. Note that this does not use the wsu:Id attribute because the schema doesn't allow for attribute extensibility.

/SecurityContextToken/Keys/SecurityTokenReference

This optional element references the shared secret of the security context.

/SecurityContextToken/Keys/{any}

This is an extensibility option to allow other types of keys/tokens to be specified.

/SecurityContextToken/@wsu:Id

This optional attribute specifies a string label for this element.

/SecurityContextToken/@{any}

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

/SecurityContextToken/{any}

This is an extensibility mechanism to allow additional elements to be used.

Security contexts, like all security tokens, can be referenced using the mechanisms described in [WS-Security](#) (the <SecurityTokenReference> element referencing the wsu:Id attribute relative to the XML base document or referencing using <wsu:Identifier> element's URI). When a token is referenced, the associated key is used. In some cases the security context might have multiple keys. To reference a specific key, the URI is constructed using the <wsu:Identifier> of the security context as a base with a local reference to the ID of the key specified using wsu:Id. For example, if the identifier of a context was <http://fabrikam123/ctx/123> then the URI for the key whose wsu:Id attribute is "key1" would be <http://fabrikam123/ctx/123#key1>.

The following sample message illustrates the use of a security context token. In this example a context has been established and the secret is known to both parties. This secret is used to sign the message body.

```
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="..." xmlns:ds="..." xmlns:wsse="...">
(003)   <S:Header>
(004)     ...
```

```

(005)     <wsse:Security>
(006)         <wsse:SecurityContextToken wsu:Id="MyID"
(007)             <wsu:Identifier>uuid:...</wsu:Identifier>
(008)         </wsse:SecurityContextToken>
(009)         <ds:Signature>
(010)             ...
(011)         <ds:KeyInfo>
(012)             <wsse:SecurityTokenReference>
(013)                 <wsse:Reference URI="#MyID"/>
(014)             </wsse:SecurityTokenReference>
(016)         </ds:KeyInfo>
(016)     </ds:Signature>
(017) </wsse:Security>
(018) </S:Header>
(019) <S:Body wsu:Id="MsgBody">
(020)     <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
           QQQ
           </tru:StockSymbol>
(021) </S:Body>
(022) </S:Envelope>

```

Let's review some of the key sections of this example:

Lines (003)-(018) contain the SOAP message headers.

Lines (005)-(017) represent the `<Security>` header block. This contains the security-related information for the message.

Lines (006)-(008) specify a [security token](#) that is associated with the message. In this case it is a security context token. Line (007) specifies the unique ID of the context.

Lines (009)-(016) specify the digital signature. In this example, the signature is based on the security context (specifically the secret/key associated with the context). Line (010) represents the typical contents of an XML Digital Signature which, in this case, references the body and potentially some of the other headers expressed by line (004).

Lines (012)-(014) indicate the key that was used for the signature. In this case, it is the security context token included in the message. Line (013) provides a URI link to the Lines (006)-(008).

The body of the message is represented by Lines (020)-(021).

4. Establishing Security Context

A security context needs to be created and shared by the communicating parties before being used. This specification defines three different ways of establishing a security context among the parties of a secure communication.

Security context token created by a security token service – The context initiator asks a security token service to create a new security context token. The newly created security context token is distributed to the parties through the protocols defined here and in [WS-Trust](#). For this scenario the initiating party sends a <RequestSecurityToken> request to the token service and a <RequestSecurityTokenResponse> is returned. The response contains a <SecurityTokenReference> pointing to the new security context token and a <ProofTokenReference> pointing to the "secret" for the returned context.

Security context token created by one of the communicating parties and propagated with a message – The initiator creates a security context token and sends it to the other parties on a message using the mechanisms described in this specification and in [WS-Security](#). This model works when the sender is trusted to always create a new security context token. For this scenario the initiating party creates a security context token and issues a signed unsolicited <RequestSecurityTokenResponse> to the other party. The message contains a <SecurityTokenReference> pointing to the new security context token and a <ProofTokenReference> pointing to the "secret" for the security context token. The recipient can then choose whether or not to accept the security context token.

Security context token created through negotiation – When there is a need to negotiate among the participants on the contents of the security context token, such as the shared secret, this specification allows the parties to exchange data to establish a security context. For this scenario the initiating party sends a <RequestSecurityToken> request to the other party and a <RequestSecurityTokenResponse> is returned. It is likely that the negotiation (challenge/response) semantics described in [WS-Trust](#) will be used. Ultimately (if successful), a final response contains a <SecurityTokenReference> pointing to the new security context and a <ProofTokenReference> pointing to the "secret" for the context.

The following illustrates a request for a security context token from a security token service.

```
<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
  <S:Header>
    ...
  <Security>
    <UsernameToken wsu:Id="myToken">
      <Username>NNK</Username>
      <Nonce>FKJh...</Nonce>
      <wsu:Created>2001-10-13T09:00:00Z </wsu:Created>
    </UsernameToken>
    <ds:Signature xmlns:ds="...">
      ...
    </ds:Signature>
  </Security>
  ...
</S:Envelope>
```

```

</S:Header>
<S:Body wsu:Id="req">
  <RequestSecurityToken>
    <TokenType>wsse:SecurityContextToken</TokenType>
    <RequestType>wsse:ReqIssue</RequestType>
    <Base>
      <Reference URI="#myToken"/>
    </Base>
  </RequestSecurityToken>
</S:Body>
</S:Envelope>

<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
  <S:Header>
    ...
  </S:Header>
  <S:Body>
    <RequestSecurityTokenResponse>
      <RequestedSecurityToken>
        <wsse:SecurityContextToken>
          <wsu:Identifier>uuid:...</wsu:Identifier>
        </wsse:SecurityContextToken>
      </RequestedSecurityToken>
      <RequestedProofToken>
        <xenc:EncryptedKey Id="newProof">
          ...
        </xenc:EncryptedKey>
      </RequestedProofToken>
    </RequestSecurityTokenResponse>
  </S:Body>

```

The following illustrates propagating a context to another party.

```

<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
  <S:Header>
    ...
  </S:Header>

```

```

<S:Body>
  <RequestSecurityTokenResponse>
    <RequestedSecurityToken>
      <wsse:SecurityContextToken>
        <wsu:Identifier>uuid:...</wsu:Identifier>
      </wsse:SecurityContextToken>
    </RequestedSecurityToken>
    <RequestedProofToken>
      <xenc:EncryptedKey Id="newProof">
        ...
      </xenc:EncryptedKey>
    </RequestedProofToken>
  </RequestSecurityTokenResponse>
</S:Body>
</S:Envelope>

```

5. Deriving Keys

A security context token contains a shared secret. This secret MAY be used for signing and/or encrypting messages, but it is RECOMMENDED that derived keys be used for signing and encrypting messages associated only with the security context.

Using a common secret, parties may define different key derivations to use. For example, four keys may be derived so that two parties can sign and encrypt using separate keys. In order to keep the keys fresh, subsequent derivations may be used. We introduce the `<DerivedKeyToken>` token as a mechanism for indicating which derivation is being used within a given message.

The derived key mechanism can use different algorithms for deriving keys. The algorithm is expressed using a QName. This specification defines one such algorithm.

We use a subset of the mechanism defined for TLS in RFC 2246. Specifically, we use the P_SHA-1 function to generate a sequence of bytes that can be used to generate security keys. We refer to this algorithm as `wsse:PSHA1`.

This function is used with three values – *secret*, *label*, and *seed*. The secret is the shared secret that is exchanged (note that if two secrets were securely exchanged, possible as part of an initial exchange, they are concatenated in the order they were sent/received). The label is the concatenation of the client's label and the service's label. These labels can be discovered in each party's policy (or specifically within a `<DerivedKeyToken>` token). If either isn't specified in the policy, then a default value of "WS-SecureConversation" is used. The seed is the concatenation of nonce values that were exchanged (initiator + receiver). The nonce seed is required, so nonces MUST be exchanged. The P_SHA-1 function has two parameters – *secret* and *value*. We concatenate the *label* and the *seed* to create the *value*. That is:

```
P_SHA1 (secret, label + seed)
```

At this point, both parties can use the P_SHA-1 function to generate shared keys as needed. For this protocol, we don't define explicit derivation uses.

The <DerivedKeyToken> element is used to indicate that the key for a specific security token is generated from the function. This is so that explicit security tokens can be exchanged for scalability purposes. However, parties MUST mutually agree on specific derivations (e.g. the first 128-bits is the client's signature key, the next 128-bits in the client's encryption key, and so on). The policy presents a method for specifying this information.

Once the parties determine a shared secret to use as the basis of a key generation sequence an initial key is generated using this sequence. When a new key is required, a new <DerivedKeyToken> may be passed referencing the previously generated key. The recipient then knows to use the sequence to generate a new key, which will match that specified in the security token. If both parties pre-agree on key sequencing, then additional token exchanges are not required.

For keys derived using a shared secret from a security context, the <SecurityTokenReference> element SHOULD be used to reference the <SecurityContextToken>. Basically, a signature or encryption references a <DerivedKeyToken> in the <Security> header which, in turn, references the <SecurityContextToken>.

The syntax for <DerivedKeyToken> is as follows:

```
<DerivedKeyToken wsu:Id="..." wsse:Algorithm="...">
  <SecurityTokenReference>...</SecurityTokenReference>
  <Properties>...</Properties>
  <Generation>...</Generation>
  <Offset>...</Offset>
  <Length>...</Length>
  <Label>...</Label>
  <Nonce>...</Nonce>
</DerivedKeyToken>
```

The following describes the attributes and tags listed in the schema overview above:

/DerivedKeyToken

This specifies a key that is derived from a shared secret.

/DerivedKeyToken/@wsu:Id

This optional attribute specifies an XML ID that can be used locally to reference this element.

/DerivedKeyToken/@wsse:Algorithm

This optional attribute specifies key derivation algorithm to use. This specification predefines the `wsse:PSHA1` algorithm described above.

/DerivedKeyToken/SecurityTokenReference

This optional element is used to specify security context token, security token, or shared key/secret used for the derivation. If not specified, it is assumed that the recipient can determine the shared key from the message context.

/DerivedKeyToken/Properties

This optional element allows metadata to be associated with this derived key. For example, if the <Name> property is defined, this derived key is given a URI name which can then be used as the source for other derived keys. The <Nonce> and <Label> elements can be specified as properties and indicate the nonce and label to use for all keys derived from this key.

/DerivedKeyToken/Generation

If fixed-size keys (generations) are being generated, then this element can be used to specify which generation of the key to use. The value of this element is an unsigned long value indicating the generation number to use (beginning with zero). This element is used when fixed-size keys are used. This element MUST NOT be used if the <Offset> element is specified.

/DerivedKeyToken/Offset

If fixed-size keys are not being generated, then the <Offset> and <Length> elements indicate where in the byte stream to find the generated key. This specifies the ordering (in bytes) of the generated output. The value of this element is an unsigned long value indicating the byte position (starting at 0). For example, 0 indicates the first byte of output and 16 indicates the 17th byte of generated output. This element MUST NOT be used if the <Generation> element is specified. It should be noted that not all algorithms will support the <Offset> and <Length> elements.

/DerivedKeyToken/Length

This element specifies the length (in bytes) of the derived key. This element can be specified in conjunction with <Offset> or <Generation>. If this isn't specified, it is assumed that the recipient knows the key size to use. The value of this element is an unsigned long value indicating the size of the key in bytes (e.g., 16).

/DerivedKeyToken/Label

If specified, this element defines a label that is used in the key derivation function. If this isn't specified, it is assumed that the recipient knows the label to use. The value of this element is a string value indicating the "label" used in the key derivation algorithm. Note that once a label is used for a derivation sequence, the same label SHOULD be used for all subsequent derivations.

/DerivedKeyToken/Nonce

If specified, this element (defined in [WS-Security](#)) specifies a nonce that is used in the key derivation function. If this isn't specified, it is assumed that the recipient knows the nonce to use. Note that once a nonce is used for a derivation sequence, the same nonce SHOULD be used for all subsequent derivations.

The following example illustrates a derived key based on the 3rd generation of the shared key identified in the specified security context:

```
<DerivedKeyToken>
  <SecurityTokenReference>
    <Reference URI="../../../ctx1"/>
  </SecurityTokenReference>
  <Generation>2</Generation>
```

```
</DerivedKeyToken>
```

The following example illustrates a derived key based on the 1st generation of a key derived from an existing derived key (4th generation):

```
<DerivedKeyToken>
  <Properties>
    <Name>.../derivedKeySource</Name>
    <Label>NewLabel</Label>
    <Nonce>FHFE...</Nonce>
  </Properties>
  <Generation>3</Generation>
</DerivedKeyToken>

<DerivedKeyToken wsu:Id="newKey">
  <SecurityTokenReference>
    <Reference URI=".../derivedKeySource"/>
  </SecurityTokenReference>
  <Generation>0</Generation>
</DerivedKeyToken>
```

In the example above we have named a derived key so that other keys can be derived from it. To do this we use the `<Properties>` element name tag to assign a global name attribute. Note that in this example, the ID attribute could have been used to name the base derived key if we didn't want it to be a globally named resource. We have also include the `<Label>` and `<Nonce>` elements as metadata properties indicating how to derive sequences of this derivation.

6. Error Handling

There are many circumstances where an *error* can occur while processing security information. Errors use the SOAP Fault mechanism.

Error that occurred	Faultcode
The requested context elements are insufficient or unsupported	wsse:BadContextToken
The specified source for the derivation is unknown.	wsse:UnknownDerivationSource

7. Security Considerations

It is critical that all relevant elements of a message be included in signatures. As well, the signatures for security context establishment **MUST** include a timestamp, nonce, or sequence number depending on the degree of replay prevention required. Security

context establishment SHOULD include full policies, but MUST include supported security algorithms.

Authenticating services are susceptible to denial of service attacks. Care should be taken to mitigate such attacks as is warranted by the service.

8. Acknowledgements

This specification has been developed as a result of joint work with many individuals and teams, including:

John Brezak, Microsoft
Tony Cowan, IBM
Slava Kavsan, RSA Security
Scott Konersmann, Microsoft
Paul Leach, Microsoft
John Linn, RSA Security
Keith Stobie, Microsoft
Richard Ward, Microsoft

9. References

[KEYWORDS]

S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," [RFC 2119](#), Harvard University, March 1997

[RFC2246]

IETF Standard, "[The TLS Protocol](#)" January 1999

[SOAP]

W3C Note, "[SOAP: Simple Object Access Protocol 1.1](#)," 08 May 2000.

[URI]

T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," [RFC 2396](#), MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[WS-Policy]

"Web Services Policy Language", BEA, IBM, Microsoft, SAP, December 2002

[WS-PolicyAttachment]

"Web Services Policy Attachment Language", BEA, IBM, Microsoft, SAP, December 2002

[WS-Security]

"Web Services Security Language", IBM, Microsoft, VeriSign, April 2002.

"WS-Security Addendum", IBM, Microsoft, VeriSign, August 2002.

"WS-Security XML Tokens", IBM, Microsoft, VeriSign, August 2002.

[WS-SecurityPolicy]

"Web Services Security Policy Assertions Language", IBM, Microsoft, RSA, VeriSign, December 2002

[WS-Trust]

"Web Services Trust Language", IBM, Microsoft, RSA, VeriSign, December 2002

[XML-C14N]

W3C Candidate Recommendation, "[Canonical XML Version 1.0](#)," 26 October 2000.

[XML-Encrypt]

W3C Recommendation, "[XML Encryption Syntax and Processing](#)," 10 December 2002.

[XML-ns]

W3C Recommendation, "[Namespaces in XML](#)," 14 January 1999.

[XML- Schema1]

W3C Recommendation, "[XML Schema Part 1: Structures](#)," 2 May 2001.

[XML- Schema2]

W3C Recommendation, "[XML Schema Part 2: Datatypes](#)," 2 May 2001.

[XML-Signature]

W3C Recommendation, "[XML-Signature Syntax and Processing](#)," 12 February 2002.