Web Services Security UsernameToken Profile 1.0

OASIS Standard 200401, March 2004

Document identifier:

{WSS: SOAP Message Security }-{UsernameToken Profile }-{1.0} (Word*)* (PDF*)*

Document Location:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0

Errata Location:

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

Editors:

Anthony	Nadalin	IBM
Phil	Griffin	Individual
Chris	Kaler	Microsoft
Phillip	Hallam-Baker	VeriSign
Ronald	Monzillo	Sun

Contributors:

Gene	Thurston	AmberPoint
Frank	Siebenlist	Argonne National Lab
Merlin	Hughes	Baltimore Technologies
Irving	Reid	Baltimore Technologies
Peter	Dapkus	BEA
Hal	Lockhart	BEA
Symon	Chang	CommerceOne
Srinivas	Davanum	Computer Associates
Thomas	DeMartini	ContentGuard
Guillermo	Lao	ContentGuard
TJ	Pannu	ContentGuard
Shawn	Sharp	Cyclone Commerce
Ganesh	Vaideeswaran	Documentum

Sam	Wei	Documentum
John	Hughes	Entegrity
Tim	Moses	Entrust
Toshihiro	Nishimura	Fujitsu
Tom	Rutt	Fujitsu
Yutaka	Kudo	Hitachi
Jason	Rouault	HP
Paula	Austel	IBM
Bob	Blakley	IBM
Joel	Farrell	IBM
Satoshi	Hada	IBM
Maryann	Hondo	IBM
Michael	McIntosh	IBM
Hiroshi	Maruyama	IBM
David	Melgar	IBM
Anthony	Nadalin	IBM
Nataraj	Nagaratnam	IBM
Wayne	Vicknair	IBM
Kelvin	Lawrence	IBM (co-Chair)
Don	Flinn	Individual
Bob	Morgan	Individual
Bob	Atkinson	Microsoft
Keith	Ballinger	Microsoft
Allen	Brown	Microsoft
Paul	Cotton	Microsoft
Giovanni	Della-Libera	Microsoft
Vijay	Gajjala	Microsoft
Johannes	Klein	Microsoft
Scott	Konersmann	Microsoft
Chris	Kurt	Microsoft

Brian	LaMacchia	Microsoft
Paul	Leach	Microsoft
John	Manferdelli	Microsoft
John	Shewchuk	Microsoft
Dan	Simon	Microsoft
Hervey	Wilson	Microsoft
Chris	Kaler	Microsoft (co-Chair)
Prateek	Mishra	Netegrity
Frederick	Hirsch	Nokia
Senthil	Sengodan	Nokia
Lloyd	Burch	Novell
Ed	Reed	Novell
Charles	Knouse	Oblix
Steve	Anderson	OpenNetwork (Sec)
Vipin	Samar	Oracle
Jerry	Schwarz	Oracle
Eric	Gravengaard	Reactivity
Stuart	King	Reed Elsevier
Andrew	Nash	RSA Security
Rob	Philpott	RSA Security
Peter	Rostin	RSA Security
Martijn	de Boer	SAP
Blake	Dournaee	Sarvega
Pete	Wenzel	SeeBeyond
Jonathan	Tourzan	Sony
Yassir	Elley	Sun Microsystems
Jeff	Hodges	Sun Microsystems
Ronald	Monzillo	Sun Microsystems
Jan	Alexander	Systinet
Michael	Nguyen	The IDA of Singapore

Don	Adams	TIBCO
John	Weiland	US Navy
Phillip	Hallam-Baker	VeriSign
Mark	Hays	Verisign
Hemma	Prafullchandra	VeriSign

Abstract:

This document describes how to use the UsernameToken with the Web Services Security (WSS) specification.

Status:

This is a technical committee document submitted for consideration by the OASIS Web Services Security (WSS) technical committee. Please send comments to the editors.

If you are on the wss@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the wss-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to wss-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For patent disclosure information that may be essential to the implementation of this specification, and any offers of licensing terms, refer to the Intellectual Property Rights section of the OASIS Web Services Security Technical Committee (WSS TC) web page at http://www.oasis-open.org/committees/wss/ipr.php. General OASIS IPR information can be found at http://www.oasis-open.org/who/intellectualproperty.shtml.

Table of Contents

1 Introduction

This document describes how to use the UsernameToken with the WSS: SOAP Message Security specification [WSS]. More specifically, it describes how a web service consumer can supply a UsernameToken as a means of identifying the requestor by “username”, and optionally using a password (or shared secret, or password equivalent) to authenticate that identity to the web service producer.

This section is non-normative.

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 [RFC 2119].

When describing abstract data models, this specification uses the notational convention used by the XML Infoset. Specifically, abstract property names always appear in square brackets (e.g., [some property]).

When describing concrete XML schemas [XML-Schema], this specification uses the notational convention of WSS: SOAP Message Security. Specifically, each member of an element’s [children] or [attributes] property is described using an XPath-like [XPath] notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (`<xs:any/>`). The use of @{any} indicates the presence of an attribute wildcard (`<xs:anyAttribute/>`).

Commonly used security terms are defined in the Internet Security Glossary [SECGLO]. Readers are presumed to be familiar with the terms in this glossary as well as the definition in the Web Services Security specification.

2.2 Namespaces

Namespace URIs (of the general form "some-URI") represents some application-dependent or context-dependent URI as defined in RFC 2396 [URI]. This specification is designed to work with the general SOAP [SOAP11, SOAP12] message structure and message processing model, and should be applicable to any version of SOAP. The current SOAP 1.1 namespace URI is used herein to provide detailed examples, but there is no intention to limit the applicability of this specification to a single version of SOAP.

The namespaces used in this document are shown in the following table (note that for brevity, the examples use the prefixes listed below but do not include the URIs – those listed below are assumed).

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/

S12	http://www.w3.org/2003/05/soap-envelope
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

The URLs provided for the *wsse* and *wsu* namespaces can be used to obtain the schema files.

2.3 Acronyms and Abbreviations

The following (non-normative) table defines acronyms and abbreviations for this document.

Term	**Definition**
SHA	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
UCS	Universal Character Set
UTF8	UCS Transformation Format, 8-bit form
XML	Extensible Markup Language

3 UsernameToken Extensions

3.1 Usernames and Passwords

The <`wsse:UsernameToken`> element is introduced in the WSS: SOAP Message Security documents as a way of providing a username.

Within <`wsse:UsernameToken`> element, a <`wsse:Password`> element may be specified. Passwords of type `wsse:PasswordText and wsse:PasswordDigest` are not limited to actual passwords, although this is a common case. Any password equivalent such as a derived password or S/KEY (one time password) can be used. Having a type of `wsse:PasswordText, wsse:PasswordDigest` merely implies that the information held in the password is “in the clear”, as opposed to holding a “digest” of the information. For example, if a server does not have access to the clear text of a password but does have the hash, then the hash is considered a *password equivalent* and can be used anywhere where a "password" is indicated in this specification. It is not the intention of this specification to require that all implementations have access to clear text passwords.

Passwords of type `wsse:PasswordText and wsse:PasswordDigest` are defined as being the Base64 [XML-Schema] encoded, SHA-1 hash value, of the UTF8 encoded password (or equivalent). However, unless this digested password is sent on a secured channel or the token is encrypted, the digest offers no real additional security over use of `wsse:PasswordText and wsse:PasswordDigest`.

Two optional elements are introduced in the `<wsse:UsernameToken>` element to provide a countermeasure for replay attacks: `<wsse:Nonce>` and `<wsu:Created>`. A nonce is a random value that the sender creates to include in each UsernameToken that it sends. Although using a nonce is an effective countermeasure against replay attacks, it requires a server to maintain a cache of used nonces, consuming server resources. Combining a nonce with a creation timestamp has the advantage of allowing a server to limit the cache of nonces to a "freshness" time period, establishing an upper bound on resource requirements. If either or both of `<wsse:Nonce>` and `<wsu:Created>` are present they MUST be included in the digest value as follows:

Password_Digest = Base64 (SHA-1 (nonce + created + password))

That is, concatenate the nonce, creation timestamp, and the password (or shared secret or password equivalent), digest the combination using the SHA-1 hash algorithm, then include the Base64 encoding of that result as the password (digest). This helps obscure the password and offers a basis for preventing replay attacks. For web service producers to effectively thwart replay attacks, three counter measures are RECOMMENDED:

1. It is RECOMMENDED that web service producers reject any UsernameToken *not* using *both* nonce *and* creation timestamps.
2. It is RECOMMENDED that web service producers provide a timestamp “freshness” limitation, and that any UsernameToken with “stale” timestamps be rejected. As a guideline, a value of five minutes can be used as a minimum to detect, and thus reject, replays.
3. It is RECOMMENDED that used nonces be cached for a period at least as long as the timestamp freshness limitation period, above, and that UsernameToken with nonces that have already been used (and are thus in the cache) be rejected.

Note that the nonce is hashed using the octet sequence of its decoded value while the timestamp is hashed using the octet sequence of its UTF8 encoding as specified in the contents of the element.

Note that `wsse:PasswordDigest` can only be used if the plain text password (or password equivalent) is available to both the requestor and the recipient.

Note that the secret is put at the end of the input and not the front. This is because the output of SHA-1 is the function's complete state at the end of processing an input stream. If the input stream happened to fit neatly into the block size of the hash function, an attacker could extend the input with additional blocks and generate new/unique hash values knowing only the hash output for the original stream. If the secret is at the end of the stream, then attackers are prevented from arbitrarily extending it -- since they have to end the input stream with the password which they don't know. Similarly, if the nonce/created was put at the end, then an attacker could update the nonce to be nonce+created, and add a new created time on the end to generate a new hash.

The countermeasures above do not cover the case where the token is replayed to a different receiver. There are several (non-normative) possible approaches to counter this threat, which may be used separately or in combination. Their use requires pre-arrangement (possibly in the form of a separately published profile which introduces new password type) among the communicating parties to provide interoperability:

- including the username in the hash, to thwart cases where multiple user accounts have matching passwords (e.g. passwords based on company name)

- including the domain name in the hash, to thwart cases where the same username/password is used in multiple systems
- including some indication of the intended receiver in the hash, to thwart cases where receiving systems don't share nonce caches (e.g., two separate application clusters in the same security domain).

The following illustrates the XML syntax of this element:

```
<wsse:UsernameToken wsu:Id="Example-1">
   <wsse:Username> ... </wsse:Username>
   <wsse:Password Type="..."> ... </wsse:Password>
   <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
   <wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

The following describes the attributes and elements listed in the example above:

/wsse:UsernameToken/wsse:Password

This optional element provides password information (or equivalent such as a hash). It is RECOMMENDED that this element only be passed when a secure transport (e.g. HTTP/S) is being used or if the token itself is being encrypted.

/wsse:UsernameToken/wsse:Password/@Type

This optional URI attribute specifies the type of password being provided. The table below identifies the pre-defined types (note that the URI fragments are relative to the URI for this specification).

URI	Description
#PasswordText (default)	The actual password for the username, the password hash, or derived password or S/KEY. This type should be used when hashed password equivalents that do not rely on a nonce or creation time are used, or when a digest algorithm other than SHA1 is used.
#PasswordDigest	The digest of the password (and optionally nonce and/or creation timestame) for the username using the algorithm described above.

/wsse:UsernameToken/wsse:Password/@{any}

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

/wsse:UsernameToken/wsse:Nonce

This optional element specifies a cryptographically random nonce. Each message including a `<wsse:Nonce>` element MUST use a new nonce value in order for web service producers to detect replay attacks.

/wsse:UsernameToken/wsse:Nonce/@EncodingType

This optional attribute URI specifies the encoding type of the nonce (see the definition of `<wsse:BinarySecurityToken>` for valid values). If this attribute isn't specified then the default of Base64 encoding is used.

/wsse:UsernameToken/wsu:Created

The optional `<wsu:Created>` element specifies a timestamp used to indicate the creation time. It is defined as part of the `<wsu:Timestamp>` definition.

All compliant implementations MUST be able to process the `<wsse:UsernameToken>` element. Where the specification requires that an element be "processed" it means that the element type MUST be recognized to the extent that an appropriate error is returned if the element is not supported.

Note that `<wsse:KeyIdentifier>` and `<ds:KeyName>` elements as described in the WSS: SOAP Message Security specification are not supported in this profile.

The following example illustrates the use of this element. In this example the password is sent as clear text and therefore this message should be sent over a confidential channel:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="...">
   <S11:Header>
      ...
      <wsse:Security>
         <wsse:UsernameToken>
            <wsse:Username>Zoe</wsse:Username>
            <wsse:Password>IloveDogs</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
      ...
   </S11:Header>
   ...
</S11:Envelope>
```

The following example illustrates using a digest of the password along with a nonce and a creation timestamp:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu= "...">
   <S11:Header>
      ...
      <wsse:Security>
         <wsse:UsernameToken>
            <wsse:Username>NNK</wsse:Username>
            <wsse:Password Type="...#PasswordDigest">
               weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==
            </wsse:Password>
            <wsse:Nonce>WScqanjCEAC4mQoBE07sAQ==</wsse:Nonce>
            <wsu:Created>2003-07-16T01:24:32Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
      ...
   </S11:Header>
   ...
</S11:Envelope>
```

3.2 Token Reference

When a UsernameToken is referenced using `<wsse:SecurityTokenReference>` the `ValueType` attribute is not required. If specified, the value of `<wsse:UsernameToken>` MUST be specified.

The following encoding formats are pre-defined (note that the URI fragments are relative to the URI for this specification):

URI	Description
#UsernameToken	UsernameToken

When a UsernameToken is referenced from a `<ds:KeyInfo>` element, it can be used to derive a key for a message authentication algorithm using the password. This profile considers specific mechanisms for key derivation to be out of scope. Implementations should agree on a key derivation algorithm in order to be interoperable.

There is no definition of a KeyIdentifier for a UsernameToken. Consequently, KeyIdentifier references MUST NOT used when referring to a UsernameToken.

Similarly, there is no definition of a KeyName for a UsernameToken. Consequently, KeyName references MUST NOT be used when referring to a UsernameToken.

All references refer to the *wsu:Id* for the token.

3.3 Error Codes

Implementations may use custom error codes defined in private namespaces if needed. But it is RECOMMENDED that they use the error handling codes defined in the WSS: SOAP Message Security specification for signature, decryption, and encoding and token header errors to improve interoperability.

When using custom error codes, implementations should be careful not to introduce security vulnerabilities that may assist an attacker in the error codes returned.

4 Security Considerations

The use of the UsernameToken introduces no additional threats beyond those already identified for other types of SecurityTokens. Replay attacks can be addressed by using message timestamps, nonces, and caching, as well as other application-specific tracking mechanisms. Token ownership is verified by use of keys and man-in-the-middle attacks are generally mitigated. Transport-level security may be used to provide confidentiality and integrity of both the UsernameToken and the entire message body.

When a password (or password equivalent) in a `<UsernameToken>` is used for authentication, the password needs to be properly protected. If the underlying transport does not provide enough protection against eavesdropping, the password SHOULD be digested as described in this document. Even so, the password must be strong enough so that simple password guessing attacks will not reveal the secret from a captured message.

When a password is encrypted, in addition to the normal threats against any encryption, two password-specific threats must be considered: replay and guessing. If an attacker can impersonate a user by replaying an encrypted or hashed password, then learning the actual password is not necessary. One method of preventing replay is to use a nonce as mentioned previously. Generally it is also necessary to use a timestamp to put a ceiling on the number of previous nonces that must be stored. However, in order to be effective the nonce and timestamp must be signed. If the signature is also over the password itself, prior to encryption, then it would be a simple matter to use the signature to perform an offline guessing attack against the password. This threat can be countered in any of several ways including: don't include the password under the signature (the password will be verified later) or sign the encrypted password.

The reader should also review Section 13 of WSS: SOAP Message Security document for additional discussion on threats and possible counter-measures.

This section is non-normative.

5 References

The following are normative references:

[SECGLO] *Informational RFC 2828, "Internet Security Glossary," May 2000.*

[RFC2119] *S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard University, March 1997*

[WSS] *OASIS standard, "WSS: SOAP Message Security," TBD.*

[SOAP11] *W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000.*

[SOAP12] *W3C Working Draft**, “**SOAP Version 1.2 Part 1: Messaging Framework”, 26 June 2002.*

[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.*

[XML-Schema] *W3C Recommendation, "XML Schema Part 1: Structures,"2 May 2001.*
W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001.

[XPath] *W3C Recommendation, "XML Path Language", 16 November 1999*

The following are non-normative references included for background and related material:

[WS-Security] *OASIS,”Web Services Security: SOAP Message Security” 19 January 2004, http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0*

[XML-C14N] *W3C Recommendation, "Canonical XML Version 1.0," 15 March 2001*

[EXC-C14N] *W3C Recommendation, "Exclusive XML Canonicalization Version 1.0," 8 July 2002.*

[XML-Encrypt] *W3C Working Draft, "XML Encryption Syntax and Processing," 04 March 2002*

W3C Recommendation, “Decryption Transform for XML Signature”, 10 December 2002.

[XML-ns] *W3C Recommendation, "Namespaces in XML," 14 January 1999.*

[XML Signature] *W3C Recommendation, "XML Signature Syntax and Processing," 12 February 2002.*

[XPointer] *"XML Pointer Language (XPointer) Version 1.0, Candidate Recommendation", DeRose, Maler, Daniel, 11 September 2001.*

Appendix A. Revision History

Rev	Date	By Whom	What
Wd-1.0	2002-12-16	Phil Griffin	Initial version cloned from the WSS core specification
Wd-1.1	2003-01-26	Anthony Nadalin	Bring in line with WSS-Core Update
Wd-1.2	2003-02-23	Anthony Nadalin	Editorial Updates
Wd-1.3	2003-06-30	Anthony Nadalin	Editorial Updates
Wd-1.4	2003-08-11	Anthony Nadalin	Editorial Updates
Cd-1.5	2003-12-09	Anthony Nadalin, Chris Kaler	Editorial Updates based on Issue List #30
Cd-1.5	2003-12-15	Anthony Nadalin, Chris Kaler	Editorial Updates based on Editorial feedback
Cd-1.6	2003-12-22	Anthony Nadalin	Editorial Updates based on Editorial feedback
Cd-1.7 & 1.8	2003-12-29	Anthony Nadalin, Chris Kaler	Editorial Updates based on Editorial feedback
Cd- 1.8	2004-01-19	Anthony Nadalin, Chris Kaler	Editorial corrections for name space and document name
Cd 1.9	2004-02-17	Anthony Nadalin	Editorial corrections per Karl Best

Appendix B. Notices