Web Services Trust Language (WS-Trust)

Version 1.0
December 18, 2002

Authors

Giovanni Della-Libera, Microsoft
Brendan Dixon, Microsoft
Praerit Garg, Microsoft
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

Abstract

This specification defines extensions that build on WS-Security to request and issue security tokens and to manage trust relationships.

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-Trust by itself does not provide a complete security solution for Web services. WS-Trust 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-Trust 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 warrantees or representations regarding the specifications in any manner whatsoever.

Table of Contents

1. Overview

WS-Security defines the basic mechanisms for providing secure messaging. This specification uses these basic mechanisms and defines additional primitives and extensions for security token exchange to enable the issuance and dissemination of credentials within different trust domains.

In order to secure a communication between two parties, the two parties must exchange security credentials (either directly or indirectly). However, each party needs to determine if they can "trust" the asserted credentials of the other party.

In this specification we define extensions to WS-Security that provide:

- methods for issuing and exchanging security tokens,
- ways to establish and access the presence of trust relationships

Using these extensions, applications can engage in secure communication designed to work with the general Web Services framework, including WSDL service descriptions, UDDI businessServices and bindingTemplates, and SOAP messages.

To achieve this, this specification introduces a number of headers and elements that are used to request security tokens and manage trust relationships.

This specification defines a number of 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 goal of WS-Trust is to enable applications to construct trusted SOAP message exchanges.

This specification is intended to provide a flexible set of mechanisms that can be used to support a range of security protocols; in other words this specification intentionally does not describe explicit fixed security protocols.

As with every security protocol, significant efforts must be applied to ensure that security protocols constructed using WS-Trust are not vulnerable to a wide range of attacks.

The following topics are outside the scope of this document:

- Establishing a security context token.
- Key derivation and exchange.

1.2. Requirements

The Web services trust specification must support a wide variety of security models. The following list identifies the key driving requirements for this specification:

- Requesting and obtaining security tokens
- Managing trusts and establishing trust relationships
- Establishing and assessing trust relationships
- Password authentication

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
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
wsp	http://schemas.xmlsoap.org/ws/2002/12/policy

2.3. Schema and WSDL Files

The schema for this specification can be located at:

```
http://schemas.xmlsoap.org/ws/2002/12/secext
```

The WSDL for this specification can be located in Appendix I of this document as well as at:

http://schemas.xmlsoap.org/ws/2002/12/ws-trust.wsdl

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).

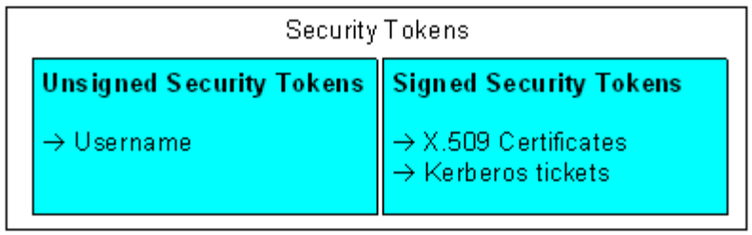
2.4. Terminology

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 makes (e.g. name, identity, key, group, privilege, capability, etc).

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

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



Proof-of-Possession – *Proof-of-possession* information is data that is used in a proof process to demonstrate that a sender is acting on behalf of a (claimed) client, based on knowledge of information that should only be known to the client. Proof-of-possession information is used to bind a client and a sender acting on behalf of a client within a security token.

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. 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 includes messages, tokens, and proofs-of-possession.

Trust Engine – The *trust engine* of a Web service is a conceptual component that evaluates the security-related aspects of a message as described in section 3 below.

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.

Trust - *Trust* is the characteristic that one entity is willing to rely upon a second entity to execute a set of actions and/or to make set of assertions about a set of subjects and/or scopes.

Direct Trust – *Direct trust* is when a relying party accepts as true all (or some subset of) the claims in the token sent by the requestor.

Direct Brokered Trust – *Direct Brokered Trust* is when one party trusts a second party who, in turn, trusts or vouches for, a third party.

Indirect Brokered Trust – *Indirect Brokered Trust* is a variation on direct brokered trust where the second party negotiates with the third party, or additional parties, to assess the trust of the third party.

Message Authentication – *Message authentication* is the process of verifying that the message received is the same as the one sent.

Origin Authentication – *Origin authentication* is the process of verifying who the sender of the message is.

Freshness - *Freshness* is the process of verifying that the message has not been replayed and is currently valid.

3. Web Services Trust Model

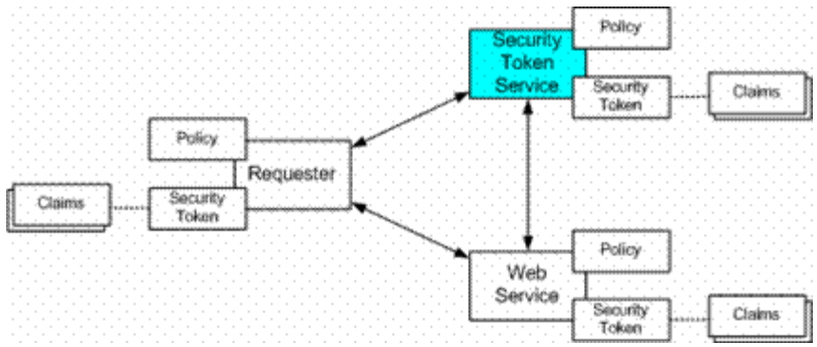
The Web service security model defined in WS-Trust is based on a process in which:

A Web service can require that an incoming message prove a set of claims (e.g., name, key, permission, capability, etc.). If a message arrives without having the required proof of claims, the service SHOULD ignore or reject the message. A service can indicate its required claims and related information in its policy as described by WS-Policy and WS-PolicyAttachment specifications.

A requester can send messages that demonstrate its ability to prove a required set of claims by associating security tokens with the messages and including signatures of the message that demonstrate proof of possession of (the contents of) the tokens.

- If the requester does not have the necessary token(s) to prove the claim, it contacts an appropriate authority and gets the tokens. These "authorities", which we refer to as *security token services*, may in turn require their own set of claims. Security token services form the basis of trust by issuing security tokens which can be used to broker trust relationships between different trust domains.
- This specification also defines a challenge response protocol. This is used by a web service for additional challenges to a requestor of the freshness and proof-of-possession information provided by the requestor.

This model is illustrated in the figure below, showing that any requester may also be a service, and that the Security Token Service is a Web service, including expressing policy and requiring security tokens.



This general messaging model – claims, policies and security tokens – subsumes and supports several more specific models such as identity-based security, access control lists, and capabilities-based security. It allows use of existing technologies such as X.509 public-key certificates, XML-based tokens, Kerberos shared-secret tickets and even password digests. The general model in combination with the WS-Security and WS-Policy primitives is sufficient to construct higher-level key exchange, authentication, policy-based access decisions, auditing, and complex trust relationships.

In summary the Web service has a policy applied to it, receives a message from a requestor that possibly includes security tokens, and may have some protection applied to it using WS-Security mechanisms. The following key steps are performed by the trust engine of a Web service (note that the order of processing is non-normative):

Verify that the claims in the token are sufficient to comply with the policy and that the message conforms to the policy.

Verify that the attributes of the claimant is proven by the signatures (performed with key associated to the necessary attributes). In brokered trust models, the signature may not verify the identity of the claimant – it may verify the identity of the intermediary, who may simply assert the identity of the claimant. The claims are either proven or not based on policy.

Verify that the issuers of the security tokens (including all related and ancestral security token) are trusted to issue the claims they have made. The trust engine may need to send tokens to a security token service in order to exchange them for other security tokens which it can use directly in its evaluation.

If these conditions are met, and the requester is authorized to perform the operation then the service can process the service request within the above specified trust model.

In this specification we define how security tokens are requested and obtained from security token services and how these services may broker trust and trust policies so that services can perform step 3.

4. Security Token Issuance, Validation and Exchange

As described above, security token services issue, validate and exchange security tokens. A requestor sends a request, and if the policy permits and the recipient's requirements are met, then the requestor receives a security token response. For this we define the `<RequestSecurityToken>` and `<RequestSecurityTokenResponse>` elements respectively. These elements are passed as the payload to specific WSDL ports (described in section 2.3) which are implemented by security token services.

The SOAP action for security token requests on a port is defined as:

```
http://schemas.xmlsoap.org/security/RequestSecurityToken
```

The SOAP action for security token responses on a port is defined as:

```
http://schemas.xmlsoap.org/security/RequestSecurityTokenResponse
```

4.1. Requesting a Security Token

The <RequestSecurityToken> element is used to request a security token. This element SHOULD be signed by the requestor, using tokens contained/referenced in the request that are relevant to the request. If using a signed request, the requestor MUST prove any required claims to the satisfaction of the security token service.

The syntax for this element is as follows:

```
<RequestSecurityToken>
    <TokenType>...</TokenType>
    <RequestType>...</RequestType>
    <Base>...</Base>
    <Supporting>...</Supporting>
</RequestSecurityToken>
```

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

/RequestSecurityToken

This is a request to have a security token issued.

/RequestSecurityToken/TokenType

This optional element describes the type of security token requested, specified as a QName. That is, the type of token that will be returned in the <RequestSecurityTokenResponse> message. If this element is not defined in a request, it is RECOMMENDED that the optional element <AppliesTo> (defined below) be used. That is, either the <TokenType> or the <AppliesTo> element SHOULD be defined within a request. If both the <TokenType> and <AppliesTo> elements are defined, the <AppliesTo> element takes precedence (for the current request only) in case the target scope requires a specific type of token.

/RequestSecurityToken/RequestType

The RequestType element is used to indicate, using a QName, the action that is being requested (e.g., wsse:Issue). The following encoding formats are pre-defined:

QName	Description
wsse:ReqIssue	Issue Security Token
wsse:ReqValidate	Validate Security Token
wsse:ReqExchange	Exchange Security Token

/RequestSecurityToken/Base

This optional element has the same type as the <SecurityTokenReference> (see WS-Security) element and references the base (primary) tokens that are used to

validate the authenticity of a request. In general, this element isn't used because signatures are provided on the request which prove the right to make the request. This element is provided in the event that multiple signatures exist over the request that don't all support the authorization of the request or if the supporting token cannot provide a signature (e.g. a <UsernameToken> token with an embedded password).

/RequestSecurityToken/Supporting

This optional element has the same type as the <SecurityTokenReference> element and references the supporting tokens that are used to authorize this request. Typically this is used to identify tokens in a certificate authority. It is not required to specify any or all supporting tokens; it is simply a hint or aid to the recipient service.

/RequestSecurityToken/{any}

This is an extensibility mechanism to allow additional elements to be added. This allows clients to include any elements which the service can use to process the token request.

/RequestSecurityToken/@{any}

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

The following is a sample request. In this example, an X.509 security token is being requested based on the security token located in the <Security> header with the ID "myToken". This token specifies a username, and a signature is placed over the request using a key derived from the password (or password equivalent), nonce and timestamp.

```
<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:Header>
    <S:Body wsu:Id="req">
       <RequestSecurityToken>
           <TokenType>wsse:X509v3</TokenType>
           <RequestType>wsse:ReqIssue</RequestType>
```

```
            <Base>
                <Reference URI="#myToken"/>
            </Base>
        </RequestSecurityToken>
    </S:Body>
</S:Envelope>
```

4.2. Returning a Security Token

The <RequestSecurityTokenResponse> element is used to return a security token.

The syntax for this element is as follows:

```
    <RequestSecurityTokenResponse>
        <TokenType>...</TokenType>
        <KeyType>...</KeyType>
        <KeySize>...</KeySize>
        <wsp:AppliesTo>...</wsp:AppliesTo>
        <RequestedSecurityToken>...</RequestedSecurityToken>
        <RequestedProofToken>...</RequestedProofToken>
    </RequestSecurityTokenResponse>
```

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

/RequestSecurityTokenResponse

This is the response to a security token request.

/RequestSecurityTokenResponse/TokenType

This optional element specifies the type of security token returned. Note that either the optional <TokenType> element or the optional <AppliesTo> element SHOULD have been specified. This MUST be provided if a token type other than the requested type is returned.

/RequestSecurityTokenResponse/KeyType

This optional element specifies the type of key used in the token (see <RequestKeyType> in the request description below).

/RequestSecurityTokenResponse/KeySize

This optional element specifies the size of the key returned (see <RequestKeySize> in the request description below).

/RequestSecurityTokenResponse/wsp:AppliesTo

This optional element specifies the scope to which this security token applies. Refer to WS-PolicyAttachment for more information.

/RequestSecurityTokenResponse/RequestedSecurityToken

This optional element is used to return the requested security token. Normally the requested security token is the contents of this element but a security token reference MAY be used instead. For example, if the requested security token is used as part of the securing of the message, then the security token is placed into the <Security> header (as described in WS-Security) and a <SecurityTokenReference>

element is placed inside of the <RequestedSecurityToken> element to reference the token in the <Security> header. This element is optional, but it is REQUIRED that at least one of <RequestedSecurityToken> or <RequestedProofToken> be returned unless there is an error.

/RequestSecurityTokenResponse/RequestedProofToken

This optional element is used to return the proof-of-possession token associated with the requested security token. Normally the proof-of-possession token is the contents of this element but a security token reference MAY be used instead. The token (or reference) is specified as the contents of this element. For example, if the proof token is used a part of the securing of the message, then it is placed in the <Security> header and a <SecurityTokenReference> element is used inside of the <RequestedProofToken> element to reference the token in the <Security> header. This element is optional, but it is REQUIRED that at least one of <RequestedSecurityToken> or <RequestedProofToken> be returned unless there is an error.

/RequestSecurityTokenResponse/{any}

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

/RequestSecurityTokenResponse/@{any}

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

The following is a sample response. In this example a pre-existing X.509v3 (looked up in a directory, followed by encoding it into a security token) security token is returned. In this example, we do not return a proof-of-possession since the client provided the public key to use (and authenticated it using corresponding private key).

```
...
<RequestSecurityTokenResponse>
    <RequestedSecurityToken>
        <BinarySecurityToken ValueType="wsse:X509v3"
                             EncodingType="wsse:Base64Binary">
            MIIEZzCCA9CgAwIBAgIQEmtJZc0...
        </BinarySecurityToken>
    </RequestedSecurityToken>
</RequestSecurityTokenResponse>
...
```

The following is another sample response. In this example a custom security token is returned along with an encrypted symmetric key for proof-of-possession.

```
...
<RequestSecurityTokenResponse>
    <RequestedSecurityToken>
        <BinarySecurityToken ValueType="x:MyToken"
                             EncodingType="wsse:Base64Binary"
                             xmlns:x="...">
            MIIEZzCCA9CgAwIBAgIQEmtJZc0...
```

```
            </BinarySecurityToken>
        </RequestedSecurityToken>
        <RequestedProofToken>
            <xenc:EncryptedKey Id="newProof">
                ...
            </xenc:EncryptedKey>
        </RequestedProofToken>
    </RequestSecurityTokenResponse>
    ...
```

4.3. Scope Requirements

This section defines extensions to the <RequestSecurityToken> element for scope requirements on the returned security token(s).

The syntax for these extension elements is as follows:

```
    <RequestSecurityToken>
        <wsp:AppliesTo>...</wsp:AppliesTo>
        <Claims>...</Claims>
    </RequestSecurityToken>
```

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

/RequestSecurityToken/wsp:AppliesTo

This optional element specifies the scope for which this security token is desired. Refer to WS-PolicyAttachment for more information. Note that either this element or the <TokenType> element SHOULD be defined in a <RequestSecurityToken> message. In the situation where BOTH fields have values, the <AppliesTo> field takes precedence.

/RequestSecurityToken/Claims

This element optionally requests a specific set of claims. In most cases, this element contains claims identified as required in a service's policy. Refer to WS-Policy for examples of how a service uses policy to specify claim requirements.

4.4. Key and Encryption Requirements

This section defines extensions to the <RequestSecurityToken> element for requesting specific types of keys or algorithms or key and algorithms as specified by a given policy in the return token(s).

The syntax for these extension elements is as follows:

```
    <RequestSecurityToken>
        <RequestKeyType>...</RequestKeyType>
        <RequestKeySize>...</RequestKeySize>
        <RequestSignatureAlgorithm>...</RequestSignatureAlgorithm>
```

```
        <RequestEncryption>...</RequestEncryption>
        <RequestProofEncryption>...</RequestProofEncryption>
        <UsePublicKey Sig=...>
            <ds:KeyInfo>...</ds:KeyInfo>
        </UsePublicKey>
        <UseKeyRef Sig=...>...</UseKeyRef>
    </RequestSecurityToken>
```

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

/RequestSecurityToken/RequestKeyType

This optional element indicates the type of key desired in the security token. The predefined values are wsse:PublicKey and wsse:SymmetricKey. Note that some security token formats have fixed key types.

/RequestSecurityToken/RequestKeySize

This optional element indicates the size of the key required. The semantics of the value are dependent on the *KeyType*. This is a request, and, as such, the requested security token is not obligated to use the requested key size. The information is provided as an indication of the desired strength of the security.

/RequestSecurityToken/RequestSignatureAlgorithm

This optional element indicates the desired signature algorithm for the returned token. This is specified as a URI indicating the algorithm (see XML Signature for typical signing algorithms).

/RequestSecurityToken/RequestEncryption

This optional element indicates that the requestor desires any returned secrets in issued security tokens to be encrypted for the specified token. That is, so that the owner of the specified token can decrypt the secret. This element has the same type as the <SecurityTokenReference> element and the contents reference the desired token. If this element isn't specified, the base token (or specialized knowledge) is used to determine how to encrypt the key.

/RequestSecurityToken/RequestProofEncryption

This optional element indicates that the requestor desires any returned secrets in proof-of-possession tokens to be encrypted for the specified token. That is, so that the owner of the specified token can decrypt the secret. This element has the same type as the <SecurityTokenReference> element and the contents reference the desired token. If this element isn't specified, the base token (or specialized knowledge) is used to determine how to encrypt the key.

/RequestSecurityToken/UsePublicKey

This optional element indicates that the requestor's public key should be used in the requested or issued security token. The key is specified using the <ds:KeyInfo> element from XML Signature.

/RequestSecurityToken/UsePublicKey/@Sig

In order to *authenticate* the public key specified, the corresponding private key MAY be used to sign this request. If specified, this optional attribute indicates the ID of the corresponding signature (by URI reference).

/RequestSecurityToken/UseKeyRef

This element has the same type as the <SecurityTokenReference> element and the contents reference the security token containing the key which should be used in the requested token. This element is meaningless and ignored if <RequestKeyType> is not defined unless a key type is implicitly known to the service.

/RequestSecurityToken/UseKeyRef/@Sig

In order to *authenticate* the public key referenced, the corresponding private key MAY be used to sign this request. If specified, this optional attribute indicates the ID of the corresponding signature (by URI reference).

In the following example the requestor is asking for a token of type "SomeTokenType" which should use 1024-bit public key technology for it's encryption needs (presumably it could optionally use symmetric keys). The requestor wants to use RSA for digital signatures and for the token service to encrypt information using a provided token. As well, the returned token should have the public key from "http://somekey".

```
<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility"
            xmlns:ds="...>
    <S:Header>
       ...
       <Security>
           <UsernameToken wsu:Id="myToken">
               <Username>NNK</wsse:Username>
               <Nonce>FKJh...</wsse:Nonce>
               <Created>2001-10-13T09:00:00Z </wsu:Created>
           </UsernameToken>
           <BinarySecurityToken wsu:Id="requestEncryptionToken"
                                ValueType="x:SomeTokenType"
                                EncodingType="wsse:Base64Binary"
                                xmlns:x="...">
                 MIIEZzCCA9CgAwIBAgIQEmtJZc0...
           </BinarySecurityToken>
           <BinarySecurityToken wsu:Id="requestProofEncryptionToken"
                                ValueType="x:SomeTokenType"
                                EncodingType="wsse:Base64Binary"
                                xmlns:x="...">
                 MIIEZzCCA9CgAwIBAgIQEmtJZc0...
           </BinarySecurityToken>
           <ds:Signature>
               ... signature over request ...
           </ds:Signature>
           <ds:Signature Id="proofSignature">
```

```
            ... signature proving requested key ...
         </ds:Signature>
      </Security>
      ...
   </S:Header>
   <S:Body wsu:Id="req">
       <RequestSecurityToken>
           <TokenType>x:SomeTokenType</TokenType>
           <RequestType>wsse:ReqIssue</RequestType>
           <Base>
               <Reference URI="#myToken"/>
           </Base>
           <RequestKeyType>wsse:PublicKey</RequestKeyType>
           <RequestKeySize>1024</RequestKeySize>
           <RequestSignatureAlgorithm>
               http://www.w3.org/2000/09/xmldsig#rsa-sha1
           </RequestSignatureAlgorithm>
           <RequestEncryption>
               <Reference URI="#requestEncryptionToken">
           </RequestEncryption>
           <RequestProofEncryption>
               <Reference URI="#requestProofEncryptionToken">
           </RequestProofEncryption>
           <UsePublicKey Sig="#proofSignature">
               <ds:KeyInfo>
                   <SecurityTokenReference>
                       <Reference URI="http://somekey"/>
                   </SecurityTokenReference>
               </ds:KeyInfo>
           </UsePublicKey>
       </RequestSecurityToken>
   </S:Body>
</S:Envelope>
```

4.5. Delegation, Forwarding, and Proxy Requirements

This section defines extensions to the <RequestSecurityToken> element for indicating delegation, forwarding, and proxy requirements on the requested security token(s).

The syntax for these extension elements is as follows:

```
    <RequestSecurityToken>
        <OnBehalfOf>...</OnBehalfOf>
        <DelegateTo>...</DelegateTo>
        <Forwardable/>
        <Delegatable/>
        <Proxiable/>
    </RequestSecurityToken>
```

/RequestSecurityToken/OnBehalfOf
: This optional element indicates that the requestor is making the request on behalf of another. The identity on whose behalf the request is being made is specified by placing a security token or <SecurityTokenReference> element within the <OnBehalfOf> element.

/RequestSecurityToken/DelegateTo
: This optional element indicates that the requested or issued token be delegated to another identity. The identity receiving the delegation is specified by placing a security token or <SecurityTokenReference> element within the <DelegateTo> element.

/RequestSecurityToken/Forwardable
: This optional element is to request a security token to be marked as "Forwardable". This allows special behaviors for type-specific token processors. In general, this is used to mark a returned token so that it can be used to obtain a token for another party or addresses based upon policy specified. The optional contents of this element MAY contain token type-specific rules and restrictions on the forwarding usage. An example of a token-specific rule would be as it applies to requesting a Kerberos ticket.

/RequestSecurityToken/Delegatable
: This optional element is to request a security token to be marked as "Delegatable". This allows special behaviors for type-specific token processors. In general, this is used to mark a returned token so that it can be delegated to another party. The optional contents of this element MAY contain token type-specific rules and restrictions on the delegation usage. An example of a token-specific rule would be as it applies to requesting a Kerberos ticket.

/RequestSecurityToken/Proxiable
: This optional element is to request a security token to be marked as "Proxiable". This allows special behaviors for type-specific token processors. In general, this is used to mark a returned token so that it can be forwarded and used by other parties or addresses (such as a proxy) than originally destined. The optional contents of this element MAY contain token type-specific rules and restrictions on the proxy usage. An example of a token-specific rule would be as it applies to requesting a Kerberos ticket.

4.6. Lifetime and Renewal Requirements

This section defines extensions to the <RequestSecurityToken> element for specifying token lifetime and renewal requirements on the requested token(s).

The WS-Security specification defines and illustrates time references in terms of the *dateTime* type defined in XML Schema. It is RECOMMENDED that all time references use this type. It is further RECOMMENDED that all references be in UTC time. Requestors and receivers SHOULD NOT rely on other applications supporting time resolution finer than milliseconds. Implementations MUST NOT generate time instants that specify leap seconds.

The syntax for these extension elements is as follows:

```
    <RequestSecurityToken>
        <AllowPostdating/>
        <Renewing Allow=... OK=.../>
        <Lifetime>
            <wsu:Created>...</wsu:Created>
            <wsu:Expires>...</wsu:Expires>
        </Lifetime>
    </RequestSecurityToken>
```

/RequestSecurityToken/AllowPostdating
: This element indicates that returned tokens should allow requests for postdated tokens.

/RequestSecurityToken/Renewing
: This optional element is used to specify renew semantics for types that support this operation.

/RequestSecurityToken/Renewing/@Allow
: This optional Boolean attribute is used to request a renewable token.

/RequestSecurityToken/Renewing/@OK
: This optional Boolean attribute is used to indicate that a renewable token is acceptable if the requested duration exceeds the limit of the issuance service.

/RequestSecurityToken/Lifetime
: This optional element is used to specify the desired valid time range for the returned security token.

/RequestSecurityToken/Lifetime/@wsu:Created
: This optional attribute specifies an absolute time representing initial validity time for the token. If this time occurs in the future then this is a request for a post-dated token. If this attribute isn't specified, then the current time is used as an initial period.

/RequestSecurityToken/Lifetime/@wsu:Expires
: This optional attribute specifies an absolute time representing the upper bound on the validity time period of the requested token. If this attribute isn't specified, then the service chooses the lifetime of the security token.

4.7. Policies

This section defines extensions to the `<RequestSecurityToken>` element for passing policies.

The syntax for these extension elements is as follows:

```
    <RequestSecurityToken>
```

```
        <wsp:Policy>...</wsp:Policy>
        <wsp:PolicyReference>...</wsp:PolicyReference>
    </RequestSecurityToken>
```

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

/RequestSecurityToken/wsp:Policy

This optional element specifies a policy (as defined in WS-Policy) which indicates desired settings for the requested token. The policy specifies defaults which can be overridden by the elements defined in the previous sections.

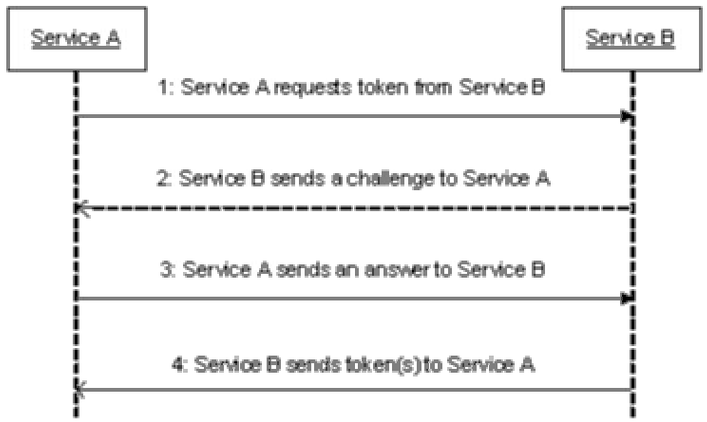
/RequestSecurityToken/wsp:PolicyReference

This optional element specifies a reference to a policy (as defined in WS-Policy) which indicates desired settings for the requested token. The policy specifies defaults which can be overridden by the elements defined in the previous sections.

4.8. Challenges

When a request for a security token is sent, the mechanisms described in WS-Security MAY be used to authenticate it. In many cases, the signed timestamps and nonces described in WS-Security are sufficient to prove freshness. However, in some cases, a service MAY choose to challenge the requestor. This section describes how challenges are issued and responded to within this framework. Please see section 4.8.2 for an example.

The process is straightforward:



A requestor sends, for example, a `<RequestSecurityToken>` message with a nonce and timestamp.

The recipient does not trust the nonce and timestamp and issues a `<RequestSecurityTokenResponse>` message with an embedded challenge.

The initiator sends a `<RequestSecurityTokenReponse>` message with an answer to the challenge.

The recipient issues a `<RequestSecurityTokenResponse>` message with the issued security token and optional proof-of-possession token.

It should be noted that the requestor might challenge the recipient in either step 1 or step 3. In which case, step 2 or step 4 contains an answer to the initiator's challenge.

Similarly, it is possible that steps 2 and 3 could iterate multiple times before the process completes (step 4).

4.8.1. Syntax

Challenge requests are issued by including an element that describes the challenge and responses contain an element describing the response. For example, signature challenges are processed using the <SignChallenge> element. The response is returned in a <SignChallengeResponse> element. Both the challenge and the response elements are specified in within the <RequestSecurityTokenResponse> element. Some forms of negotiation MAY specify challenges along with responses to challenges from the other party. It should be noted that the requestor MAY challenge the recipient in the initial request. Consequently, these elements are also allowed within a <RequestSecurityToken> element.

It is RECOMMENDED that a <Nonce> element and <Timestamp> header be included in challenges as a way to ensure freshness of the challenge. Other types of challenges MAY also be included. For example, a <Policy> element may be used to negotiate desired policy behaviors of both parties. Multiple challenges and responses MAY be included.

The syntax of these elements is as follows:

```
<SignChallenge>
    <Challenge ...>...</ Challenge>
    <SecurityTokenReference>...</SecurityTokenReference>
</SignChallenge>

<SignChallengeResponse>
    <SecurityTokenReference>...</SecurityTokenReference>
    <ds:Signature>...</ds:Signature>
</SignChallengeResponse>

<SecurityTokenReference ...>
    ...
</SecurityTokenReference>

<BinaryNegotiation ValueType="...">
    ...
</BinaryNegotiation>
```

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

.../SignChallenge

This optional element describes specifies a challenge that requires the other party to sign a specified set of information.

.../SignChallenge/Challenge

This required element describes what parts of the challenge must be signed as part of a valid response. This element uses the same type as the `<Integrity>` assertion described in WS-SecurityPolicy.

.../SignChallenge/SecurityTokenReference

This optional element is used to reference security tokens passed as part of the negotiation process.

…/SignChallenge/{any}

This is an extensibility mechanism to allow additional negotiation types to be used.

…/SignChallenge/@{any}

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

.../SignChallengeResponse

This optional element describes a response to a challenge that requires the signing of a specified set of information.

.../SignChallengeResponse/ds:Signature

This optional element is the XML Signature element for attaching a signature. This is specified by a party as a way of proving their answer to a signature challenge. Note that there MAY be multiple signatures if multiple tokens are being used to authenticate.

.../SignChallengeResponse/SecurityTokenReference

This optional element is used to reference security tokens passed as part of the negotiation process.

…/SignChallengeResponse/{any}

This is an extensibility mechanism to allow additional negotiation types to be used.

…/SignChallengeResponse/@{any}

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

.../SecurityTokenReference

This optional element is used to reference security tokens passed as part of the negotiation process. In some cases the negotiation takes place using existing security token formats (for example, legacy binary formats). In such cases, this element is used to reference the token. In other cases the negotiation data may be arbitrary binary blobs and the `<BinaryNegotiation>` element SHOULD be used.

.../BinaryNegotiation

This optional element is used for a security negotiation that involves exchange binary blobs as part of an existing negotiation protocol. The contents of this element are blob-type-specific and are encoded using Base64 (unless otherwise specified).

.../BinaryNegotiation/@ValueType

This required attribute specifies a QName to identify the type of negotiation (and the value space of the blob – the element's contents).

.../BinaryNegotiation/@EncodingType

This required attribute specifies a QName to identify the encoding format (if different from Base64) of negotiation blob.

…/BinaryNegotiation/@{any}

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

4.8.2. Example

Here is an example exchange. In this example a service requests an X.509 certificate using another X.509 certificate for authentication.

```
<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
    <S:Header>
       ...
       <Security>
           <BinarySecurityToken
             wsu:Id="myToken"
             ValueType="wsse:X509v3"
             EncodingType="wsse:Base64Binary">
             MIIEZzCCA9CgAwIBAgIQEmtJZc0...
           </BinarySecurityToken>
           <ds:Signature xmlns:ds="...">
               ...
           </ds:Signature>
       </Security>
       ...
    </S:Header>
    <S:Body>
       <RequestSecurityToken>
           <TokenType>wsse:X509v3</TokenType>
           <RequestType>wsse:ReqIssue</RequestType>
           <Base>
               <Reference URI="#myToken"/>
           </Base>
       </RequestSecurityToken>
    </S:Body>
</S:Envelope>
```

The recipient service doesn't trust the sender's timestamp (or one wasn't specified) and issues a challenge:

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

```
      <Security>
          <BinarySecurityToken
            wsu:Id="myToken2"
            ValueType="wsse:X509v3"
            EncodingType="wsse:Base64Binary">
            DFJHuedsujfnrnv45JZc0...
          </BinarySecurityToken>
          <ds:Signature xmlns:ds="...">
              ...
          </ds:Signature>
          <Nonce>FKJh...</Nonce>
      </Security>
      ...
   </S:Header>
   <S:Body>
   <RequestSecurityTokenResponse>
       <SignChallenge>
           <Challenge>
             <MessageParts xmlns:wsse=".../secext">
                 GetInfosetForNode(GetHeader(/)/wsse:Security/wsse:Nonce)
             </MessageParts>
           </Challenge>
       </SignChallenge>
   </RequestSecurityTokenResponse>
   </S:Body>
</S:Envelope>
```

The initiator receives the recipient's challenge and issues a response:

```
<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
   <S:Header>
      ...
      <Security>
          <BinarySecurityToken
            wsu:Id="myToken"
            ValueType="wsse:X509v3"
            EncodingType="wsse:Base64Binary">
            MIIEZzCCA9CgAwIBAgIQEmtJZc0...
```

```
            </BinarySecurityToken>
            <ds:Signature xmlns:ds="...">
                ...
            </ds:Signature>
        </Security>
        ...
    </S:Header>
    <S:Body>
        <RequestSecurityTokenResponse>
            <SignChallengeResponse>
                <ds:Signature xmlns:ds="...">
                    ...
                </ds:Signature>
            </SignChallengeResponse>
        </RequestSecurityTokenResponse>
    </S:Body>
</S:Envelope>
```

The recipient validates the initiator's signature responding to the challenge and issues the requested token(s).

```
<S:Envelope xmlns:S="..." xmlns=".../secext" xmlns:wsu=".../utility">
    <S:Header>
        ...
        <Security>
            <BinarySecurityToken
              wsu:Id="myToken2"
              ValueType="wsse:X509v3"
              EncodingType="wsse:Base64Binary">
              DFJHuedsujfnrnv45JZc0...
            </BinarySecurityToken>
            <ds:Signature xmlns:ds="...">
                ...
            </ds:Signature>
        </Security>
        ...
    </S:Header>
```

```
    <S:Body>
    <RequestSecurityTokenResponse>
        <RequestedSecurityToken>
           <BinarySecurityToken
             ValueType=”wsse:X509v3”
             EncodingType="wsse:Base64Binary">
             MIIEZzCCA9CgAwIBAgIQEmtJZc0...
           </BinarySecurityToken>
        </RequestedSecurityToken>
        <RequestedProofToken>
           <ds:KeyInfo xmlns:ds="...">
               <ds:KeyValue>
                   ...
               </ds:KeyValue>
           </ds:KeyInfo>
        </RequestedProofToken>
    </RequestSecurityTokenResponse>
    </S:Body>
</S:Envelope>
```

5. Management of Trust

There are several different models that can be used to bootstrap trust for a service. This section presents several different mechanisms that could be used. These mechanisms are non-normative and are not required in any way. That is, services are free to bootstrap trust using any mechanism.

Fixed trust roots – The simplest mechanism is where the recipient has a fixed set of trust relationships. It will then evaluate all requests to determine if they contain security tokens from one of the trusted roots.

Trust hierarchies – Building on the trust roots mechanism, a service may choose to allow hierarchies of trust so long as the trust chain eventually leads to one of the known trust roots. In some cases the recipient may require the sender to provide the full hierarchy. In other cases, the recipient may be able to dynamically fetch the tokens for the hierarchy from a token store.

Authentication service – Another approach is to use an authentication service. This can essentially be thought of as a fixed trust root where the recipient only trusts the authentication service. Consequently, the recipient forwards tokens to the authentication service which replies with an authoritative statement (perhaps a separate token or a signed document) attesting to the authentication.

6. Models for Trust Assessment

In this section we discuss the two methods of assessing the presence of a trust relationship. These methods depend on whether the assessment is based on information from within a message flow (in-band) or if it is external to a message flow (out-of-band).

6.1. In-band

As part of a message flow, a request may be made of a security token service to exchange a security token (or some proof) of one form for another. The exchange request can be made either by a requestor or by another party on the requestor's behalf. If the security token service trusts the provided security token (because, for example, it trusts the issuing authority of the provided security token), and the request can prove possession of that security token, then the exchange is processed by the security token service. This is an example of an in-band direct trust relationship. In the case of a delegated request (one in which another party provides the request on behalf of the requester rather than the requester presenting it themselves) the security token service generating the new token may not need to trust the authority which issued the original token provided by the client since it does trust the security token service that is engaging in the exchange for a new security token. The basis of the trust is the relationship between the two security token services.

To perform this exchange the `RequestSecurityToken` and `RequestSecurityTokenResponse` elements for requesting and returning security tokens (as previously described) are used.

6.2. Out-of-Band

An administrator or other trusted authority may designate that all tokens of a certain type are trusted. The security token service maintains this as a trust axiom and can communicate this to trust engines to make their own trust decisions (or revoke it later), or the security token service may provide this function as a service to trusting services.

7. Password-Based Key Derivation

In some scenarios, authentication will be based on usernames and passwords as described in WS-Security. If a secure transport isn't used, then the password MUST NOT be transmitted in the clear. Instead, it is RECOMMEDED that a password be used as a shared secret from which a key is derived and used to "sign" digests, thereby proving ownership of the username token.

It is possible that multiple key derivation algorithms could be used. Consequently, we define the *wsse:Algorithm* attribute which can be specified on the `<UsernameToken>` element. This QName attribute specifies the algorithm to use.

In this specification we define an initial algorithm, `wsse:PWDPSHA1`, which is the default algorithm if one isn't specified.

To compute the signing key using `wsse:PWDPSHA1`, a subset of the mechanism defined for TLS in RFC 2246 MUST be used. Specifically, we use the P_SHA-1 function to generate a sequence of bytes that can be used to generate security keys.

The *secret* is the password, the *label* is the client label (optionally specified in policies), and the seed is the *nonce* value specified by the client (a `<Nonce>` element is required). If such an element is specified in the `<Username>` element it is used, otherwise the value

from the <Security> element is used. If a <wsu:Created> time element is specified in the <UsernameToken>, then it is also used. If this message is part of a shared context with another party, then the label is the concatenation of the client and server labels and the seed is the concatenation of the client and server nonces. The nonce is processed as a binary octet stream and the timestamp as a UTF-8 encoded string.

```
P_SHA1 (password, label + nonce + timestamp)
```

At this point, both parties can use the P_SHA-1 function to generate shared keys as needed.

If the recipient has a public key, the <ds:SignatureValue> element of the signature SHOULD be encrypted for the recipient to prevent certain types of attacks.

Services MAY specify how they derive keys from passwords in their policy.

8. 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 request was invalid or malformed	wsse:InvalidRequest
Authentication Failed	wsse:FailedAuthentication
The specified request failed	wsse:RequestFailed
Security token has been revoked	wsse:InvalidSecurityToken
Insufficient Digest Elements	wsse:AuthenticationBadElements
The specified RequestSecurityToken is not understood.	wsse:BadRequest
The request data is out-of-date	wsse:ExpiredData
The requested time range is invalid or unsupported	wsse:InvalidTimeRange

9. Security Considerations

The following statements about signatures and signing apply to messages sent on unsecured channels.

It is critical that all the security-sensitive message elements MUST be included in the scope of the message signature. As well, the signatures for conversation authentication MUST include a timestamp, nonce, or sequence number depending on the degree of replay prevention required as described in WS-Security and the addendum. Also, conversation establishment SHOULD include the policy so that supported algorithms and algorithm priorities can be validated.

It is REQUIRED that security token issuance messages be signed to prevent tampering. If a public key is provided, the request SHOULD be signed by the corresponding private key to prove ownership. As well, additional steps should be taken to eliminate replay attacks (refer to WS-Security and the addendum for additional information). Similarly, all token references SHOULD be signed to prevent any tampering.

Security token requests are susceptible to denial of service attacks. Care should be taken to mitigate such attacks as is warranted by the service.

For security, tokens containing a symmetric key or a password should only be sent to parties who have a need to know that key or password.

For privacy, tokens containing personal information (either in the claims, or indirectly by identifying who is currently communicating with whom) should only be sent according to the privacy policies governing these data at the respective organizations.

10. Acknowledgements

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

Bob Blakley, IBM
John Brezak, Microsoft
Tony Cowan, IBM
Satoshi Hada, IBM
Heather Hinton, IBM
Slava Kavsan, RSA Security
Scott Konersmann, Microsoft
John Linn, RSA Security
Paul Leach, Microsoft
Keith Stobie, Microsoft
Richard Ward, Microsoft

11. 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 Framework", BEA, IBM, Microsoft, SAP, December 2002

[WS-PolicyAssertion]
"Web Services Policy Assertions 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

[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 Candidate Recommendation, "XML-Signature Syntax and Processing," 31 October 2000.

[X509]
S. Santesson, et al,"Internet X.509 Public Key Infrastructure Qualified Certificates Profile,"

[Kerberos]
J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993, http://www.ietf.org/rfc/rfc1510.txt .

Appendix I – WSDL

The WSDL below does not fully capture all the possible message exchange patterns, but captures the typical message exchange pattern as described in this document.

```
<?xml version="1.0"?>
<wsdl:definitions
    targetNamespace="http://schemas.xmlsoap.org/ws/2002/12/secext"
    xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
<!-- this is the WS-I BP-compliant way to import a schema -->
      <wsdl:types>
          <xs:schema>
              <xs:import
                 namespace="http://schemas.xmlsoap.org/ws/2002/12/secext"
                 schemaLocation="secext.xsd"/>
```

```
          </xs:schema>
      </wsdl:types>

<!-- WS-Trust defines exactly two GEDs, here they are -->
      <wsdl:message name="RequestSecurityTokenMsg">
         <wsdl:part name="request" element="wsse:RequestSecurityToken" />
      </wsdl:message>
      <wsdl:message name="RequestSecurityTokenResponseMsg">
         <wsdl:part name="response"
                    element="wsse:RequestSecurityTokenResponse" />
      </wsdl:message>

<!-- This portType models the full request/response the Security Token
Service: -->

        <wsdl:portType name="WSSecurityRequester">
              <wsdl:operation name="SecurityTokenResponse">
                    <wsdl:input
                        message="wsse:RequestSecurityTokenResponseMsg"/>
              </wsdl:operation>
              <wsdl:operation name="Challenge">
                    <wsdl:input
                        message="wsse:RequestSecurityTokenResponseMsg"/>
                    <wsdl:output
                        message="wsse:RequestSecurityTokenResponseMsg"/>
              </wsdl:operation>
        </wsdl:portType>

<!-- These portTypes model the individual message exchanges -->

      <wsdl:portType name="SecurityTokenRequestService">
              <wsdl:operation name="RequestSecurityToken">
                    <wsdl:input message="wsse:RequestSecurityTokenMsg"/>
              </wsdl:operation>
        </wsdl:portType>

        <wsdl:portType name="SecurityTokenService">
```

```
            <wsdl:operation name="RequestSecurityToken">
                  <wsdl:input message="wsse:RequestSecurityTokenMsg"/>
                  <wsdl:output
                      message="wsse:RequestSecurityTokenResponseMsg"/>
            </wsdl:operation>
       </wsdl:portType>
</wsdl:definitions>
```