
Web-Services Security Quality of Protection

1. Introduction (Non-normative)

1.1. Problem statement

Using the WS Security Specification, service end-points have a standard means for securing SOAP messages using XML Signature and XML Encryption. However, the WSS specification does not provide a means for a Web-service consumer and provider to negotiate how the SOAP messages used in the exchange have to be protected in order to successfully invoke the service. In this paper, a technique for negotiating a mutually-acceptable security policy based on WSDL is proposed.

The term security policy is used in this context to mean: "a statement of the requirements for protecting arguments in a WS API, including:

- how actors are to be authenticated, using what mechanisms and with what parameter value ranges,
- which SOAP messages, elements and attachments are to be encrypted, for what individual recipients, recipient roles or keys, using what algorithms and key sizes,
- which SOAP messages, elements and attachments are to be integrity protected, using what mechanisms, with which algorithms and key sizes and
- what initial parameter values are used in the signature validation procedure, including what keys or authorities are trusted directly".

This is a relatively restrictive use of the term "security policy". A more comprehensive definition addresses such requirements as:

- privacy (retention period, intended usage, further disclosure),
- authorization (additional qualifications the service consumer must demonstrate in order to successfully access the Web service) and
- non-repudiation (requirements for notarization and time-stamping).

These topics are touched on later in the paper.

In some situations (notably RPC-style service invocation), Web service interactions are conducted in a persistent session. In such cases, the provider's security policy may indicate the requirements for authenticity, integrity and confidentiality of the session.

In other situations (notably document-style service invocation), messages must be protected in isolation. In these cases, while the service provider may declare a policy, the service consumer actually creates the service request. So, the consumer actually chooses which security operations to apply to the messages. Therefore, we need a way for the consumer to discover the service provider's policy and choose a set of security operations that are consistent with both its own and the service-provider's security policies.

Multi-cast environments introduce unique requirements that we do not attempt to address here.

1.2. Document outline

In this document we describe an approach to solving the stated problem.

Section 1.3 describes how this problem is solved today and points out the shortcomings of the existing solutions. Section 1.3.1 describes the proposed approach in outline. Section 1.5 describes a number of potential process models for applying the proposed approach. Section 2 describes the data model of the proposed elements, and Section 3 describes their schema. Section 4 contains an example WSDL <definitions> element to illustrate the approach. Section 5 lists the identifiers used in the specification.

Section 6 discusses some security and privacy considerations. Section 7 examines whether the approach described here can serve as a basis for other types of security policy. Section 8 lists some issues that remain to be resolved. Section 9 lists the individuals who contributed to the preparation of this paper. Appendix A contains the schema listing for the proposed elements. Appendix B contains the proposed schema for the WSDL definition of a secured SOAP interface. Appendix C contains the proposed schema for the WSS header contents for conveying the service consumer's policy for responses in a service request.

1.3. Existing solutions

Three main solutions to this problem already exist:

1. the consumer and provider agree, in some unspecified manner, which protection mechanisms to apply to which elements and messages,
2. the provider is capable of accepting, and willing to accept, a broad range of mechanisms, and the consumer chooses which mechanism to use and which elements and messages to apply them to, and
3. the CPP/CPA approach described by ebXML.

The first approach is probably satisfactory when the provider and consumer are governed by the same policy authority. This situation exists most commonly where the provider and consumer are in applications owned and operated by the same corporate entity.

The second approach is probably satisfactory in circumstances where the security policy may be set entirely by the consumer.

Neither of these solutions is satisfactory in a federated environment, where both provider and consumer independently define their own security policies, and messages have to be exchanged in a way that satisfies both policies.

1.3.1. The ebXML CPA Negotiation Protocol

The ebXML Collaboration-Protocol Profile and Agreement Specification describes a technique for deriving (amongst other things) mutually-acceptable quality of protection for exchanges between a service provider and a service consumer.

As described here, the WSDL document of a Web-service would include a security policy description representing the types of security operations that are required and supported by the Web-service for its SOAP message exchanges with consumers. However, since Web-service consumers may themselves be implemented as a Web-service, both the consumer and provider of the service may have a security policy defined in their WSDL documents. This implies that there is a need for the Web-service consumers and providers to agree on the applied security policy to be used to protect the SOAP message exchange between them.

The process of agreement of the effective security policy to be used by two Web-services is part of the domain of negotiation protocols. The OASIS ebXML CPP/CPA group has an effort currently underway to develop a negotiation protocol to generate a CPA using the collaborative business party profile that includes information about the security, transport and message exchange characteristics of the two parties involved in the ebXML/SOAP message exchange. The ebXML Negotiation protocol scope includes negotiation of the contents of a CPA including security policies. These negotiations could be applicable for both long-term relationships and single message exchanges between a Web-service consumer and provider. Furthermore, the specific parts of a Web-services security policy negotiation protocol that may be useful to address relate to the messages that are publicly exchanged between two web services that may have their

own WSDL security policy descriptions defined. These negotiation messages may have interoperability issues, and therefore, the documents that are exchanged between the two Web-services, as part of the negotiation messages, need to be standardized.

The following are two of the issues pertaining to use of the ebXML CPA negotiation protocol that need to be factored in as part of the development of a run-time model for security policy processing.

1) Status of ebXML CPA Negotiation Protocol. The OASIS ebXML negotiation protocol is a work in progress. Hence, there are some dependencies with the OASIS CPP/CPA TC.

2) WSDL Mapping of CPP/CPA. CPP/CPA is basically a super-set of WSDL, since WSDL provides binding and service information, while CPP additionally provides organizational information about the participating Web-service parties (e.g., role of organization) in context of a particular service, as well as error-handling and other failure scenarios.

Finally, during any standards effort related to this paper, a decision may need to be made whether negotiation protocols should be out-of-scope for the first version or whether a simplistic model in which a service provider controls the effective security policy used in the SOAP message exchanges is sufficient

1.4. Basic approach

We describe an approach that allows a service-consumer to discover and retrieve a service-provider's security policy for service requests, and allows a service-consumer to send its own security policy for service responses to the service-provider, as illustrated in Figure 1. The service consumer combines its own policy for service requests with that of the service provider to obtain the "applied security policy" for requests, which specifies the set of security operations that the consumer must perform on the request. The combining takes place in such a way that the applied security policy is consistent with both the consumer's and provider's security policies. Likewise, the service provider combines its own policy for responses with that of the consumer, to obtain the applied security policy for responses.

We propose to introduce a new WSDL binding to support the publication of the security policy in the case that a provider offers a secured interface. Specifically, elements called <SecurityMechanisms> and <SecurityServices> are associated with message definitions in the service's WSDL instance.

In addition, we specify a WSS header for conveying the consumer's policy for service responses using the same element definitions.

The <SecurityMechanisms> element describes a set of security mechanism, which may be applied to one or more nodes of the SOAP document. Examples of security mechanisms include: transport level security (e.g. SSL 2.0, SSL 3.0, TLS, etc.) and message level security (e.g. XML Digital Signatures). Additionally, parameters of a security mechanism may be specified in the element. Examples of a mechanism parameters include the minimum size of the data-encryption key. A reference to the mechanism definition is contained in the <SecurityServices> element, and for each mechanism the nodes to which it must be applied are listed.

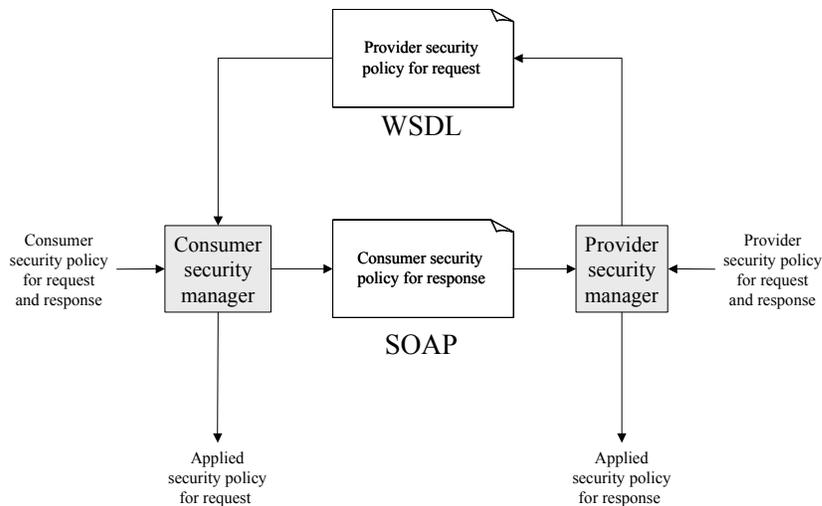


Figure 1 - Basic approach

1.5. Process models

This paper does not specify an architectural model or protocols. It only specifies the syntax and semantics of data structures. However, this section describes an architectural model and protocols to help illustrate the approach.

Two process models are described:

- The deployment-time model and
- The run-time model.

In the deployment-time model, the provider's policy for requests is discovered by the service-consumer at the time at which the consumer class is deployed. And in the run-time model, the service-provider security policy is discovered by the service-consumer at run-time. In both cases, the service-provider receives the consumer's policy for responses at the time the service is invoked.

1.5.1. Deployment-time model

Security policy is set and managed by an authority separately from the activities of the class developer. The security policy manager must be capable of including the <SecurityMechanisms> and <SecurityServices> elements in the provider class's WSDL instance, in accordance with the WSDL secure-soap binding provided in Appendix B.

One further model taxonomy applies. The security service could be implemented "out-of-line" (see Section 1.5.2) or "in-line" (see Section 1.5.3).

1.5.2. Out-of-line model

The out-of-line model is illustrated in Figure 2.

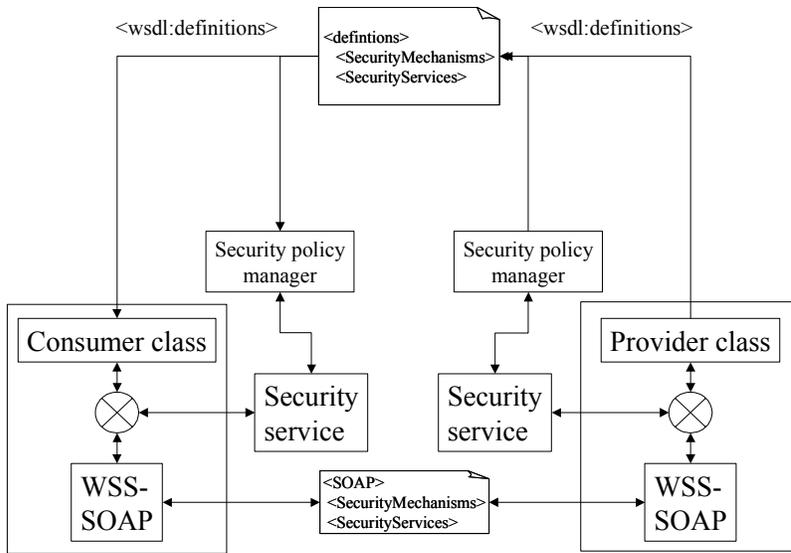


Figure 2 - Out-of-line model

And the sequence of exchanges is illustrated in Figure 3 and Figure 4.

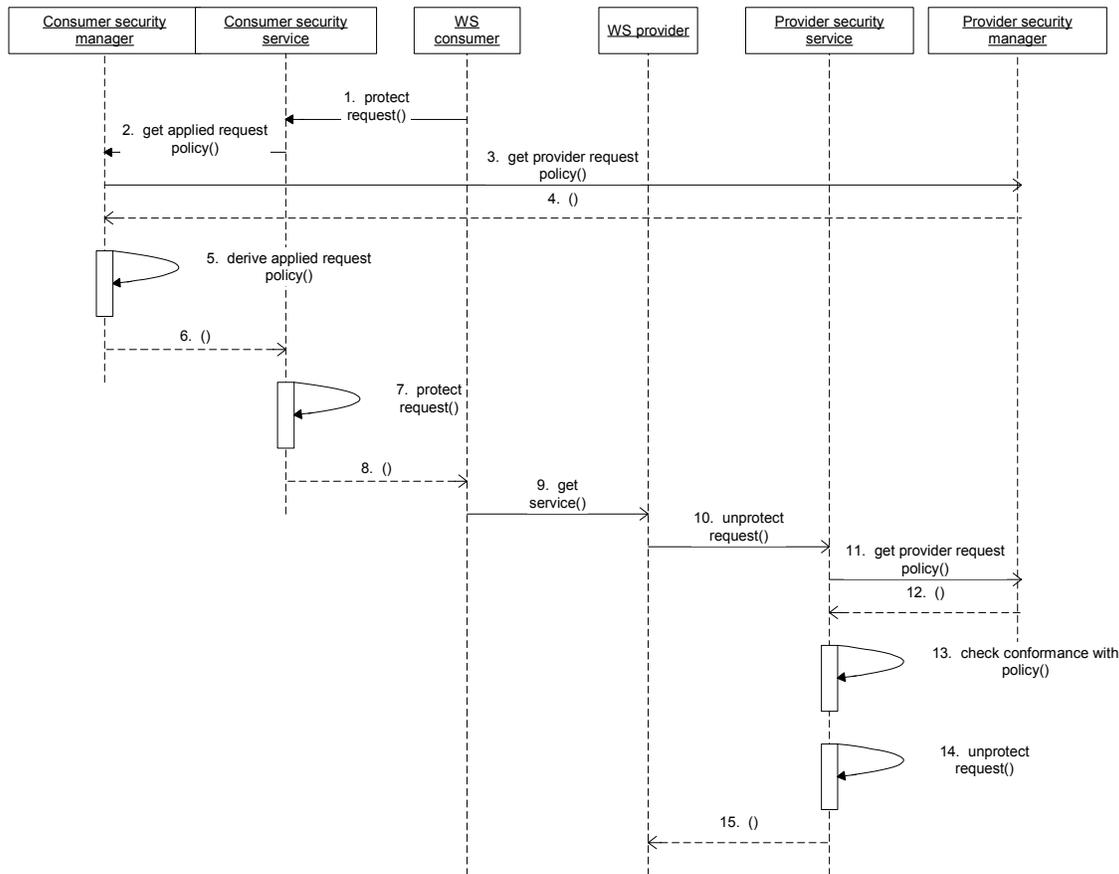


Figure 3 - Request sequence

In steps 1-8, the consumer's security service protects the service request for transmission to the service provider in step 9. In steps 10-15, the provider's security service unprotects the request for presentation to the service interface.

In steps 2-6, the consumer's security service requests the applied security policy for the service request type from its security manager. The applied security policy is the set of security operations that are actually to be performed on a service request of that type.

In steps 7 and 8, the consumer's security service protects the request, in accordance with the applied security policy, and return it to the consumer.

In steps 3 and 4, the consumer's security manager obtains the provider's security policy for the request type, embedded in the service's WSDL <definitions> element. In step 5, the consumer's security manager derives the applied security policy from both the consumer's and provider's security policies for the request type, in such a way that it is consistent with both policies. In step 6, it returns the applied policy to the consumer's security service.

In steps 11-13, the provider's security service verifies that the request conforms with its security policy for the request type. In steps 11 and 12, the provider's security service requests the provider's security policy for the request from the provider's security manager.

As a byproduct of this process, in steps 6-10, the consumer's security policy for the corresponding response is transmitted to the provider's security service. In step 6, the consumer's security manager sends the consumer's security policy for the corresponding response to the consumer's security service. The consumer's security service formats the policy as a WSS header (see Appendix C) and returns it to the consumer in step 8. The consumer appends the header and sends it with the service request to the service provider in step 9. The service provider then sends it to the provider's security service in step 10. The provider's security service must retain the consumer's security policy for the response type for use in protecting the response.

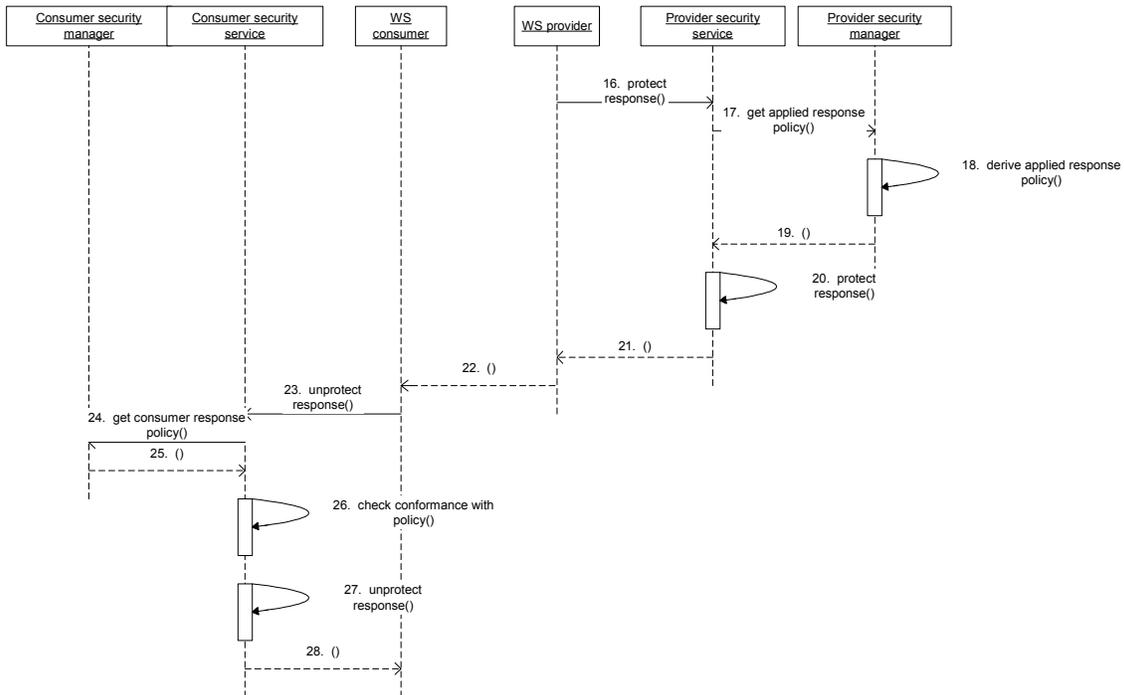


Figure 4 - Response sequence

In steps 16-21, the provider's security service protects the service response for transmission to the service consumer in step 22. In steps 23-28, the consumer's security service unprotects the response for return to the service consumer.

In steps 17-19, the provider's security service requests the applied security policy for the service response type from its security manager. The provider's security service includes the consumer's security policy for the response type in the request. In step 18, the provider's security manager derives the applied security policy from both the consumer's and provider's security policies for this response type.

In step 20, the provider's security service protects the response, in accordance with the applied security policy.

In steps 24-26, the consumer's security service verifies that the response conforms with its security policy for this response type. In steps 24 and 25, the consumer's security service requests the consumer's security policy for this response type from the consumer's security manager. And in step 27, it unprotects the response.

1.5.3. In-line model

The in-line model works similarly, with the exception that the containers hosting the consumer and provider classes emit a SOAP message, which is intercepted by the security service. The consumer and provider classes could provide the `<SecurityMechanisms>` and `<SecurityServices>` elements to their security services, in a WSS header, with the security service module identified as the target role. Alternatively, the security service could obtain the `<SecurityMechanisms>` and `<SecurityServices>` elements directly on its own (see Figure 5).

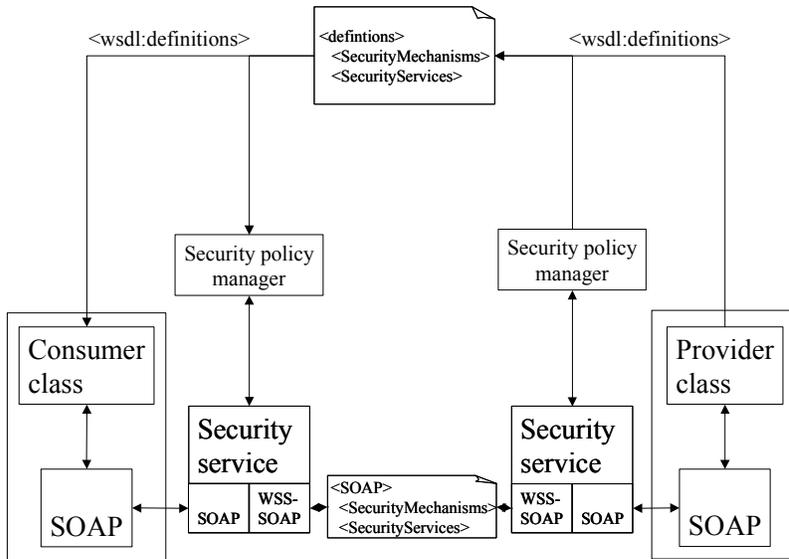


Figure 5 - In-line model

1.5.4. Run-time model

The only thing that distinguishes the run-time model from the deployment-time model is that the service's WSDL instance and its `<SecurityMechanisms>` and `<SecurityServices>` elements are retrieved at the time the service is invoked. This approach may be useful if the consumer may obtain the service from any one of a number of providers, each with differing security policies. It may also be useful if the `<SecurityMechanisms>` and `<SecurityServices>` elements are generated automatically, using a separate security-policy negotiation protocol.

While this approach ensures that the `<SecurityMechanisms>` and `<SecurityServices>` elements are fresh, since they are always retrieved dynamically, the impact on performance may be unacceptable in many situations.

2. Data model (non-normative)

The data model of the `<SecurityMechanisms>` element is shown in Figure 6.

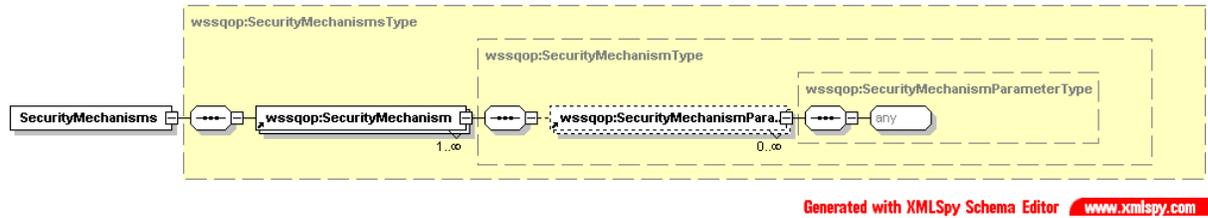


Figure 6 – <SecurityMechanisms> data model

The data model of the <SecurityServices> element is shown in Figure 7.

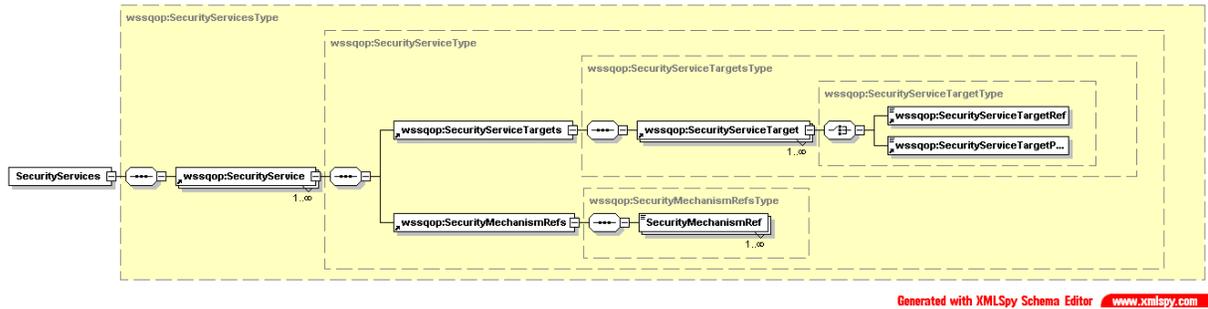


Figure 7 – <SecurityServices> data model

3. Functional specification (normative with the exception of schema fragments and examples)

3.1. <SecurityMechanisms> element

The <SecurityMechanisms> element is a container for security mechanism definitions. The order of the security mechanism definitions is not significant. Mechanisms are linked to message elements in the security service definitions (see Section 3.5).

```
<xs:element name="SecurityMechanisms" type="wssqop:SecurityMechanismsType"/>
<xs:complexType name="SecurityMechanismsType">
  <xs:sequence>
    <xs:element ref="wssqop:SecurityMechanism" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Here is an example of its use:

```
<secure-soap:SecurityMechanisms>
  <wssqop:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCe"
SecurityMechanismId="wssqop.xml-dsig-v1.0">
  ...
  </wssqop:SecurityMechanism>
  <wssqop:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCf"
SecurityMechanismId="wssqop.xml-dsig-v1.0">
  ...
  </wssqop:SecurityMechanism>
  <wssqop:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCg"
SecurityMechanismId="wssqop.xml-dsig-v1.0">
  ...
  </wssqop:SecurityMechanism>
</secure-soap:SecurityMechanisms>
```

It contains a list of mechanism definitions. In this example each mechanism is a variant of XML Digital Signature.

3.2. <SecurityMechanism> element

The <SecurityMechanism> element identifies a single security operation. It contains a SecurityMechanismRef attribute by which security services may reference the operation and link them to message elements. It also contains a SecurityMechanismId attribute that identifies the security operation and zero or more <SecurityMechanismParameter> elements containing parameters for the operation. There is no significance to the order of the parameters. Standard values for the SecurityMechanismId can be found in Section 5.3.

```
<xs:element name="SecurityMechanism" type="wssqp:SecurityMechanismType"/>
<xs:complexType name="SecurityMechanismType">
  <xs:sequence>
    <xs:element ref="wssqp:SecurityMechanismParameter" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="SecurityMechanismRef" type="xs:NCName"/>
  <xs:attribute name="SecurityMechanismId" type="xs:anyURI"/>
</xs:complexType>
```

Note: Some security operations may not require any parameters.

Here is an example of its use.

```
<wssqp:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCe"
SecurityMechanismId="wssqp:xml-dsig-v1.0">
  <wssqp:SecurityMechanismParameter SecurityParameterId="wssqp:digital-signature-algorithm">
    <xs:anyURI>ds#rsa-sha1</xs:anyURI>
  </wssqp:SecurityMechanismParameter>
  <wssqp:SecurityMechanismParameter SecurityParameterId="wssqp:minimum-key-size">
    <xs:nonNegativeInteger>2048</xs:nonNegativeInteger>
  </wssqp:SecurityMechanismParameter>
</wssqp:SecurityMechanism>
```

It defines a variant of the XML Digital Signature mechanism that uses 2048-bit RSA with the SHA-1 digest algorithm.

3.3. <SecurityMechanismParameter> element

A <SecurityMechanismParameter> element contains a single parameter for a security operation. It contains a single SecurityParameterId attribute and a single security parameter value. Standard values for the SecurityParameterId can be found in Section 5.4 and standard values for certain parameters can be found in Section 5.5.

```
<xs:element name="SecurityMechanismParameter" type="wssqp:SecurityMechanismParameterType"/>
<xs:complexType name="SecurityMechanismParameterType">
  <xs:sequence>
    <xs:any/>
  </xs:sequence>
  <xs:attribute name="SecurityParameterId" type="xs:anyURI"/>
</xs:complexType>
```

Here is an example of its use.

```
<wssqp:SecurityMechanismParameter SecurityParameterId="wssqp:digital-signature-algorithm">
  <xs:anyURI>ds#rsa-sha1</xs:anyURI>
</wssqp:SecurityMechanismParameter>
```

It illustrates the use of a digital-signature algorithm identifier, in this case RSA with the SHA-1 digest algorithm.

3.4. <SecurityServices> element

The <SecurityServices> element is a container for security service definitions. All the security services identified by a <SecurityServices> element MUST be applied to the request message by the service-

consumer, and to the response message by the service-provider, in the order in which they appear in the <wsdl:definitions> element, and they must be removed from the request message by the service-provider, and from the response message by the service-consumer in the reverse order.

```
<xs:element name="SecurityServices" type="wssqop:SecurityServicesType"/>
<xs:complexType name="SecurityServicesType">
  <xs:sequence>
    <xs:element ref="wssqop:SecurityService" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Here is an example of its use.

```
<secure-soap:SecurityServices>
  <wssqop:SecurityService SecurityServiceId="wssqop:message-integrity">
...
  </wssqop:SecurityService>
  <wssqop:SecurityService SecurityServiceId="wssqop:message-integrity">
...
  </wssqop:SecurityService>
</secure-soap:SecurityServices>
```

It illustrates the use of two security services, both for message-integrity. Both services are required to be applied. Contents of the service elements indicate which mechanism is to be used and which elements are to be protected with the mechanism.

3.5. <SecurityService> element

Each <SecurityService> element defines a set of payload nodes and mechanisms that apply to those nodes. A <SecurityService> element contains a <SecurityServiceTargets> element that identifies the payload nodes and a <SecurityMechanismRefs> element that identifies the security operations to be applied to the payload nodes. It also contains a SecurityServiceId attribute that identifies the security service. Standard values for the SecurityServiceId can be found in Section 5.2.

```
<xs:element name="SecurityService" type="wssqop:SecurityServiceType"/>
<xs:complexType name="SecurityServiceType">
  <xs:sequence>
    <xs:element ref="wssqop:SecurityServiceTargets"/>
    <xs:element ref="wssqop:SecurityMechanismRefs"/>
  </xs:sequence>
  <xs:attribute name="SecurityServiceId" type="xs:anyURI"/>
</xs:complexType>
```

Here is an example of its use.

```
<wssqop:SecurityService SecurityServiceId="wssqop:message-integrity">
  <wssqop:SecurityServiceTargets>
...
  </wssqop:SecurityServiceTargets>
  <wssqop:SecurityMechanismRefs>
...
  </wssqop:SecurityMechanismRefs>
</wssqop:SecurityService>
```

It illustrates the application of the message-integrity service, with sub-elements to indicate to which nodes the service is to be applied and which mechanisms are to be used.

3.6. <SecurityServiceTargets> element

A <SecurityServiceTargets> element identifies a set of payload nodes. It contains one or more <SecurityServiceTarget> elements, each of which identifies a single payload node. The originator of a message under this policy MUST apply the corresponding security mechanisms to (at least) all the identified payload nodes.

```

<xs:element name="SecurityServiceTargets" type="wssqop:SecurityServiceTargetsType"/>
<xs:complexType name="SecurityServiceTargetsType">
  <xs:sequence>
    <xs:element ref="wssqop:SecurityServiceTarget" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

Here is an example of its use.

```

<wssqop:SecurityServiceTargets>
  <wssqop:SecurityServiceTarget>
...
  </wssqop:SecurityServiceTarget>
</wssqop:SecurityServiceTargets>

```

It illustrates a single target definition.

3.7. <SecurityServiceTarget> element

A <SecurityServiceTarget> element identifies a payload node to which a mechanism is to be applied and (optionally) assigns an identifier to the resulting element, so that subsequent operations can be applied to the result, if necessary.

```

<xs:element name="SecurityServiceTarget" type="wssqop:SecurityServiceTargetType"/>
<xs:complexType name="SecurityServiceTargetType">
  <xs:choice>
    <xs:element ref="wssqop:SecurityServiceTargetRef"/>
    <xs:element ref="wssqop:SecurityServiceTargetPath"/>
  </xs:choice>
  <xs:attribute name="SecurityServiceTargetId" type="xs:string" use="optional"/>
</xs:complexType>
<xs:element name="SecurityServiceTargetRef" type="xs:string"/>
<xs:element name="SecurityServiceTargetPath" type="xs:anyURI"/>

```

The <SecurityServiceTargetRef> child element carries location information about the payload. Consequently, this could contain a URI representing a MIME content or href. The <SecurityServiceTargetPath> element may contain an XPath/Xpointer value to identify the payload. The <SecurityServiceTargetRef> element could also point to a remotely located payload component that may need specific protection schemes. The <SecurityServiceTargetId> attribute is used to identify the element that may be created as a result of applying the security operation, so that it can be identified as the target of subsequent security operations in their <SecurityServiceTargetRef> elements. The value of this attribute may not actually be attached to any element produced by the security operation. It may simply be used internal to the security service that performs the security operations.

Here is an example of its use.

```

<wssqop:SecurityServiceTarget>
  <wssqop:SecurityServiceTargetRef>quote</wssqop:SecurityServiceTargetRef>
</wssqop:SecurityServiceTarget>

```

It illustrates the definition of the message “quote” as the target of the security service.

3.8. <SecurityMechanismRefs> element

The <SecurityMechanismRefs> element is a container for references to security mechanisms. If multiple <SecurityMechanismRef> elements are present, then one and only one of the identified mechanisms SHALL be applied to the message. A reference is valid if the value of the <SecurityMechanismRef> element in the <SecurityMechanismRefs> element is equal to the SecurityMechanismRef attribute in the <SecurityMechanism> element (see Section 3.2).

```

<xs:element name="SecurityMechanismRefs" type="wssqop:SecurityMechanismRefsType"/>

```

```

<xs:complexType name="SecurityMechanismRefsType">
  <xs:sequence>
    <xs:element name="SecurityMechanismRef" type="xs:QName" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

Here is an example of its use.

```

<wssqop:SecurityMechanismRefs>
  <wssqop:SecurityMechanismRef>IWijxQjUrcXBYoCf</wssqop:SecurityMechanismRef>
</wssqop:SecurityMechanismRefs>

```

It illustrates a reference to the mechanism whose MechanismRef attribute is "IWijxQjUrcXBYoCf" that defines an XML Digital Signature mechanism (see Section 3.1).

4. Example (non-normative)

This section contains an example illustrating a WSDL document of a Web-service that offers two operations: buying stocks and getting stock quotes. The WSDL description includes security service and mechanism definitions that express the characteristics of security operations applicable to the request and response SOAP messages to and from the Web-service.

We show here the two ways of including the security policy elements in a WSDL document. The first involves adding security mechanism definitions as a direct child of the <wsdl:definitions> element. The second involves adding security service definitions as an extensibility element of the <wsdl:binding> element of a WSDL document. Both approaches require new processing behaviour for WSDL processors.

To support the binding extension approach, the WSDL example uses a new SOAP binding called WSDL secure-soap binding, which is described in Appendix B. Furthermore, the WSDL document example is for RPC-style request-response SOAP messages. However, the WSDL secure-soap binding could also be applied for document-style SOAP messages. Lastly, the WSDL example here uses the standard HTTP transport binding, i.e., <http://schemas.xmlsoap.org/soap/http>, for transmission of SOAP messages. Other transport bindings may also be used as part of the wsdl:binding component.

The <wssqop:SecurityMechanism> associated with the getStockQuote operation requires no authentication, but the message is signed by the Web-service provider to provide integrity. The <wssqop:SecurityMechanism> associated with the buyStock operation requires that the input and output messages be signed. The security-related parts of the WSDL document are highlighted in the example below.

```

[01] <?xml version="1.0" encoding="UTF-8"?>
[02] <definitions name="StockQuoteService" targetNamespace="http://qop.oasis.net"
  xmlns:tns="http://example.net/Stocks.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:secure-soap="http://schemas.xmlsoap.org/wsdl/secure-soap/" xmlns:soap-
  env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wssqop="urn:oasis:names:tc:wssqop:1.0" xmlns:wsse="http://wsse.oasis.net"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
[03] http://schemas.xmlsoap.org/wsdl/" xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/secure-
  soap/ C:\dev\schemas\wsdl\_sec.xsd">
[04]   <secure-soap:SecurityMechanisms>
[05]     <wssqop:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCe"
  SecurityMechanismId="wssqop:xml-dsig-v1.0">
[06]       <wssqop:SecurityMechanismParameter SecurityParameterId="wssqop:digital-signature-
  algorithm">
[07]         <xs:anyURI>ds#rsa-sha1</xs:anyURI>
[08]       </wssqop:SecurityMechanismParameter>
[09]     <wssqop:SecurityMechanismParameter SecurityParameterId="wssqop:minimum-key-

```

```

size">
[10]     <xs:nonNegativeInteger>2048</xs:nonNegativeInteger>
[11]     </wssqop:SecurityMechanismParameter>
[12]     </wssqop:SecurityMechanism>
[13]     <wssqop:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCf"
SecurityMechanismId="wssqop.xml-dsig-v1.0">
[14]     <wssqop:SecurityMechanismParameter SecurityParameterId="wssqop:digital-signature-
algorithm">
[15]         <xs:anyURI>ds#dsa-sha1</xs:anyURI>
[16]         </wssqop:SecurityMechanismParameter>
[17]         <wssqop:SecurityMechanismParameter SecurityParameterId="wssqop:minimum-key-
size">
[18]             <xs:nonNegativeInteger>1024</xs:nonNegativeInteger>
[19]             </wssqop:SecurityMechanismParameter>
[20]             </wssqop:SecurityMechanism>
[21]             <wssqop:SecurityMechanism SecurityMechanismRef="IWijxQjUrcXBYoCg"
SecurityMechanismId="wssqop.xml-dsig-v1.0">
[22]                 <wssqop:SecurityMechanismParameter SecurityParameterId="wssqop:trust-anchor">
[23]                     <ds:keyInfo>
[24]                         <ds:RetrievalMethod>http://example.xyz-
bank.com/certificate</ds:RetrievalMethod>
[25]                         </ds:keyInfo>
[26]                     </wssqop:SecurityMechanismParameter>
[27]                 </wssqop:SecurityMechanism>
[28]             </secure-soap:SecurityMechanisms>
[29]     <message name="stockQuoteRequest">
[30]         <part name="stock" type="xsd:string"/>
[31]     </message>
[32]     <message name="stockQuoteResponse">
[33]         <part name="quote" type="xsd:string"/>
[34]         <part name="date" type="xsd:date"/>
[35]     </message>
[36]     <message name="buyStockRequest">
[37]         <part name="stock" type="xsd:string"/>
[38]         <part name="number" type="xsd:integer"/>
[39]         <part name="limit" type="xsd:string"/>
[40]         <part name="validUntil" type="xsd:date"/>
[41]     </message>
[42]     <message name="buyStockResponse">
[43]         <part name="confirmationNumber" type="xsd:string"/>
[44]     </message>
[45]     <portType name="StockPortType">
[46]         <operation name="getStockQuote">
[47]             <input message="tns:stockQuoteRequest" name="stock"/>
[48]             <output message="tns:stockQuoteResponse" name="quote"/>
[49]         </operation>
[50]         <operation name="buyStock">
[51]             <input message="tns:buyStockRequest" name="stockToBuy"/>
[52]             <output message="tns:buyStockResponse" name="confirmationNumber"/>
[53]         </operation>
[54]     </portType>
[55]     <binding name="StockBinding" type="tns:StockPortType">
[56]         <secure-soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
[57]         <secure-soap:operation name="buyStock">
[58]             <secure-soap:operation soapAction=""/>
[59]         <secure-soap:SecurityServices>
[60]             <wssqop:SecurityService SecurityServiceId="wssqop:message-integrity">
[61]                 <wssqop:SecurityServiceTargets>
[62]                     <!--classify operation for input/output/fault-->
[63]                     <wssqop:SecurityServiceTarget>
[64]                         <!-- perform operation on message:sign-->
[65]                         <!--message refers to the message as it is defined in portType/operation-->
[65]                         <wssqop:SecurityServiceTargetRef>stockToBuy
[66]                     </wssqop:SecurityServiceTarget>
[67]                 </wssqop:SecurityServiceTargets>

```

```

[68]         <!--These are references to mechanism, which need to be applied!-->
[69]         <wssqop:SecurityMechanismRefs>
[70]             <wssqop:SecurityMechanismRef>IWijxQjUrcXBYoCe
</wssqop:SecurityMechanismRef>
[71]         </wssqop:SecurityMechanismRefs>
[72]         </wssqop:SecurityService>
[73]         <wssqop:SecurityService SecurityServiceId="wssqop:message-integrity">
[74]             <wssqop:SecurityServiceTargets>
[75]                 <!--classify operation for input/output/fault-->
[76]                 <wssqop:SecurityServiceTarget>
[77]                     <!-- perform operation on message:sign-->
[78]                     <wssqop:SecurityServiceTargetRef
type="message">confirmationNumber</wssqop:SecurityServiceTargetRef>
[79]                 </wssqop:SecurityServiceTarget>
[80]             </wssqop:SecurityServiceTargets>
[81]             <!--These are references to mechanism, which need to be applied!-->
[82]             <wssqop:SecurityMechanismRefs>
[83]                 <wssqop:SecurityMechanismRef>IWijxQjUrcXBYoCf
</wssqop:SecurityMechanismRef>
[84]             </wssqop:SecurityMechanismRefs>
[85]         </wssqop:SecurityService>
[86]     </secure-soap:SecurityServices>
[87]     <secure-soap:input name=" stockToBuy ">
[88]         <secure-soap:body use="encoded" namespace="urn:stock"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
[89]     </secure-soap:input>
[90]     <secure-soap:output name="confirmationNumber">
[91]         <secure-soap:body use="encoded" namespace="urn:stock"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
[92]     </secure-soap:output>
[93] </secure-soap:operation>
[94] <secure-soap:operation name="getStockQuote">
[95]     <secure-soap:operation soapAction="" />
[96]     <secure-soap:SecurityServices>
[97]         <wssqop:SecurityService SecurityServiceId="wssqop:message-integrity">
[98]             <wssqop:SecurityServiceTargets>
[99]                 <!--classify operation for input/output/fault-->
[100]                 <wssqop:SecurityServiceTarget>
[101]                     <!-- perform operation on message:sign-->
[102]                     <wssqop:SecurityServiceTargetRef>quote
</wssqop:SecurityServiceTargetRef>
[103]                 </wssqop:SecurityServiceTarget>
[104]             </wssqop:SecurityServiceTargets>
[105]             <!--These are references to mechanism, which need to be applied!-->
[106]             <wssqop:SecurityMechanismRefs>
[107]                 <wssqop:SecurityMechanismRef>IWijxQjUrcXBYoCf
</wssqop:SecurityMechanismRef>
[108]             </wssqop:SecurityMechanismRefs>
[109]         </wssqop:SecurityService>
[110]     </secure-soap:SecurityServices>
[111]     <secure-soap:input name="stock">
[112]         <secure-soap:body use="encoded" namespace="urn:stock"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
[113]     </secure-soap:input>
[114]     <secure-soap:output name="quote">
[115]         <secure-soap:body use="encoded" namespace="urn:stock"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
[116]     </secure-soap:output>
[117] </secure-soap:operation>
[118] </binding>
[119] <service name="StockService">
[120]     <documentation>Provides access to the stocks</documentation>
[121]     <port name="StockPort" binding="tns:StockBinding">
[122]         <secure-soap:address location="https://a.b.c:80/soap/stocks"/>
[123]     </port>
[124] </service>

```

[125] </definitions>

Lines [04] - [28] contain the <SecurityMechanisms> element for the parent <definition> element. It contains three security mechanism definitions in lines [05] – [12], lines [13] – [20] and lines [21] - [27], respectively. The security mechanisms have references “IWijxQjUrcXBYoCe”, etc..

Lines [59] – [86] contain the <SecurityServices> element for the getStockQuote “stockToBuy” ([65]) input message and “confirmationNumber” output message ([78]). It identifies a two security mechanisms, either one of which is acceptable for protecting the message. See lines [70] and [83]. Line [65] and [78] identify the target of the security service.

Lines [96] – [110] contain the <SecurityServices> element for the “quote” output message. It identifies a single security service that will be applied by the service-provider on the response message, see line [107]. Line [102] identifies the message to which the security services will be applied.

5. Identifiers (normative)

This section defines the names, types and value ranges for the identifiers used in the specification.

5.1. Declaration

xmlns:wssqop="urn:oasis:names:tc:wssqop:1.0:identifier"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"

5.2. Service identifiers

Service name	Service id	Type	Value range
Transport confidentiality	wssqop:transport-confidentiality	Set of Mechanism uris	subset-of(wssqop:ssl-v2.0 wssqop:ssl-v3.0 wssqop:tls-v1.0 wssqop:kerberos-v5.0)
Transport integrity	wssqop:transport-integrity	Set of Mechanism uris	subset-of(wssqop:ssl-v2.0 wssqop:ssl-v3.0 wssqop:tls-v1.0 wssqop:kerberos-v5.0)
Message confidentiality	wssqop:message-confidentiality	Set of Mechanism uris	subset-of(xenc#rsa-1_5 wssqop:xml-enc-v1.0)
Message integrity	wssqop:message-integrity	Set of Mechanism uris	subset-of(xenc#rsa-1_5 wssqop:xml-dsig-v1.0)
Originator authenticity	wssqop:originator-authenticity	Set of Mechanism uris	subset-of(wssqop:username-password wssqop:xml-dsig)

5.3. Mechanism identifiers

Mechanism name	Mechanism id	Type	Value range
SSL V2.0	wssqop:ssl-v2.0	Set of Parameter uris	all-of(wssqop:digest-algorithm wssqop:key-management-algorithm wssqop:data-encryption-algorithm wssqop:trust-anchor wssqop:minimum-key-size)
SSL V3.0	wssqop:ssl-v3.0	Set of Parameter uris	all-of(wssqop:digest-algorithm wssqop:key-management-algorithm wssqop:data-encryption-algorithm wssqop:trust-anchor wssqop:minimum-key-size)

TLS V1.0	wssqop:tls-v1.0	Set of Parameter uris	all-of(wssqop:digest-algorithm wssqop:key-management-algorithm wssqop:data-encryption-algorithm wssqop:trust-anchor wssqop:minimum-key-size)
Kerberos V5.0	wssqop:kerberos-v5.0	Set of Parameter uris	all-of(wssqop:digest-algorithm wssqop:key-management-algorithm wssqop:data-encryption-algorithm wssqop:minimum-key-size)
PKCS#1 V1.5	xenc#rsa-1_5	Set of Parameter uris	all-of(wssqop:digest-algorithm wssqop:key-management-algorithm wssqop:data-encryption-algorithm wssqop:trust-anchor wssqop:minimum-key-size wssqop:content-type)
XML Encryption V1.0	wssqop:xml-enc-v1.0	Set of Parameter uris	all-of(wssqop:key-management-algorithm wssqop:data-encryption-algorithm wssqop:minimum-key-size wssqop:token-type)
XML Digital signature V1.0	wssqop:xml-dsig-v1.0	Set of Parameter uris	all-of(wssqop:digest-algorithm wssqop:key-management-algorithm wssqop:digital-signature wssqop:trust-anchor wssqop:token-type)

5.4. Parameter identifiers

Parameter name	Parameter id	Type	Value range
Digest algorithm	wssqop:digest-algorithm	Set of Value uris	subset-of(wssqop:md5 ds#sha1)
Message authentication	wssqop:message-authentication	Set of Value uris	subset-of(ds#hmac-sha1)
Key management algorithm	wssqop:key-management-algorithm	Set of Value uris	subset-of(ds#rsa-sha1 xenc#dh)
Digital signature	wssqop:digital-signature	Set of Value uris	subset-of(ds#rsa-sha1 ds#dsa-sha1)
Data-encryption algorithm	wssqop:data-encryption-algorithm	Set of Value uris	subset-of(wssqop:rc4 xenc#tripleDES-cbc xenc#aes128-cbc)
Minimum key size	wssqop:minimum-key-size	Non-negative integer	
Content type	wssqop:content-type	Urn	subset-of(wssqop:signed-data wssqop:enveloped-data wssqop:signed-and-enveloped-data)
Trust anchor	wssqop:trust-anchor	ds:keyInfo	Set of self-signed certificates
Token type	wssqop:token-type	Set of Value uris	subset-of(wssqop:x.509-v3.0 wssqop:saml-v1.0)

5.5. Parameter values

Name	Parameter value
MD5	wssqop:md5
SHA-1	Ds#sha1
HMAC-SHA-1	Ds#hmac-sha1
RSA	Ds#rsa-sha1
Diffie-Hellman	xenc#dh
DSA	Ds#dsa-sha1
RC-4	wssqop:rc4
Triple DES	xenc#tripleDES-cbc
AES	wssqop:aes128-cbc
Signed data	wssqop:signed-data
Enveloped data	wssqop:enveloped-data
Signed and enveloped data	wssqop:signed-and-enveloped-data
X.509 V3	wssqop:x.509-v3.0
SAML V1.0	wssqop:saml-v1.0

6. Security and privacy considerations (non-normative)

If the <SecurityServices> element were to be distributed without integrity/authenticity protection, an adversary could successfully substitute a weaker policy, thereby tricking the consumer into protecting its messages weakly. This situation is avoided if each domain enforces its own security policy. Alternatively, or additionally, the <SecurityServices> element could be signed and verified.

Naturally, despite having stipulated a policy, an end-point may receive a message that violates the policy. The policy SHOULD specify how to address this situation. The message may be processed anyway; it could be argued that, in the case of confidentiality, any damage is already done and the situation cannot be mitigated by a refusal to process the message. However, in order to avert a repeat of this failure, it is RECOMMENDED that a fault be returned, and the message should not be processed.

7. Authorization, Privacy and non-repudiation (non-normative)

The approach outlined above is designed to address the policy requirements of WSS. It needs to be understood whether it can also serve as a suitable basis for the other aspects of security policy.

7.1. Authorization

It has been argued that quality of protection and authorization lie on the continuum of security policies, and that it is artificial to draw a sharp distinction between the two. It has also been argued that service-providers will be unwilling to publish authorization policy for an interface, because that information assists adversaries to discover and exploit vulnerabilities in the interface.

If we accept that quality of protection and authorization do indeed lie on a continuum, then solutions put forward for quality of protection should be able to satisfy the requirements of authorization as well. Only experience will help us to understand whether there is a need for service-providers to publish their authorization policies. So, the safe course of action is to assume that such a requirement exists, and if this premise turns out to be wrong, then the outcome is simply that the facility will not be used in practice.

We propose as a working definition that authorization policy places limits on the acceptable *values of, and relationships amongst, the identities and attributes of actors* (also known as subjects), *parts* (also known as resources) and *operations* (also known as actions) as well as attributes of the *environment* of a service-provider interface. Note: this definition casts message-integrity as a QoP service when it used for accountability, but as an authorization service when it is used for access control.

The language for expressing these limitations should have the following characteristics: it must be capable of expressing requirements for multiple subjects, it must include mathematical operators and it must be capable of expressing actions that must be performed in conjunction with access to the interface.

A number of well-established languages exist with these characteristics. So, the only remaining question is how statements in the chosen language should be attached to a service interface definition. Here is an example, based on XACML.

Name	Value
SecurityServiceId	“wssqop:authorization”
SecurityMechanismId	“wssqop:xacml”
SecurityParameterId	“wssqop:condition”
Parameter value	an <xacml:Policy> element

Perhaps, it will, one day, prove possible to generate a workflow definition directly from an interface’s authorization policy, so that a qualified service-consumer will be able to automatically select the sequence of steps required to create a conformant service request. But, that is in the future.

7.2. Privacy

Perhaps the main way in which a privacy policy differs from an authorization policy is that privacy-policy statements apply to *other* interfaces of the service-provider’s business process rather than to the one being accessed by the service-consumer. It is through these other interfaces that information supplied by the service-consumer will be further disclosed.

In addition to the characteristics cited above, in order to support privacy requirements, the chosen policy language must be capable of combining authorization policies written by the service-consumer and the service-provider. Similarly, perhaps it will become possible to define the service-provider’s workflow directly from the combined policies of the service-consumer and provider, so that the provider’s business process can be automatically adapted to conform with the combined policy.

Privacy policy, then, could be attached to a service-provider’s interface definition in exactly the same way that authorization policy can be.

7.3. Non-repudiation

WSS is intended to protect messages during their transfer between service end-points. Upon arrival at its destination, it is assumed that all associated protection mechanisms may be replaced with protection mechanisms that are more appropriate to the service-execution environment.

The topic of non-repudiation deals with long-term protection of the payload between arm-length corporate entities (because it contemplates a dispute-resolution phase that follows service invocation by potentially a lengthy period of time). Therefore, a service’s non-repudiation requirements may apply to the payload, rather than to the envelope. Nevertheless, such requirements could be stated in the service’s interface definition.

Long-term integrity and authenticity protection depends upon ensuring that private keys were used within their non-revoked, non-expired, validity period and that suitable procedures govern key management. These types of procedure are commonly simply aggregated under an identifier. Unless the service-provider obtains the required timestamp, the interface definition may have to indicate that a timestamp must be provided by the consumer. Otherwise, no special requirements are identified for the interface definition.

7.4. Conclusion

It appears that the framework described here *is* suitable for expressing requirements for authorization, privacy and non-repudiation. More work is required for fuller definition of the solution in these areas.

8. Open Issues (non-normative)

The following issues remain to be addressed.

- Make use of SOAP 1.2 “features”.
- Consider providing a means for type-checking security parameters.
- Provide a document-style example.
- Provide an example involving a protected attachment.
- Provide an example using the <SecurityServiceTargetPath> element.
- Confirm consistency of the approach with that adopted by WSDL 1.2.
- Clarify any IPR issues in relation to ebXML CPP/CPA and other applicable techniques.

9. Contributors (non-normative)

The following individuals contributed to the preparation of this paper.

Zahid Ahmed, Commerce One Inc.

Martijn de Boer, SAP AG

Monica Martin, Drake Certivo Inc.

Prateek Mishra, Netegrity Inc.

Dale Moberg, Cyclone Commerce

Ron Monzillo, Sun Microsystems

Tim Moses, Entrust Inc.

Rob Philpott, RSA Security Inc

Gene Thurston, AmberPoint Inc.

Appendix A – Schema (normative)

Following is the schema for the <SecurityServices> and <SecurityMechanisms> elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:oasis:names:tc:wssqop:1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wssqop="urn:oasis:names:tc:wssqop:1.0" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
  <xs:element name="SecurityMechanisms" type="wssqop:SecurityMechanismsType"/>
  <xs:complexType name="SecurityMechanismsType">
    <xs:sequence>
      <xs:element ref="wssqop:SecurityMechanism" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="SecurityMechanism" type="wssqop:SecurityMechanismType"/>
  <xs:complexType name="SecurityMechanismType">
    <xs:sequence>
      <xs:element ref="wssqop:SecurityMechanismParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="SecurityMechanismRef" type="xs:NCName"/>
    <xs:attribute name="SecurityMechanismId" type="xs:anyURI"/>
  </xs:complexType>
  <xs:element name="SecurityMechanismParameter" type="wssqop:SecurityMechanismParameterType"/>
  <xs:complexType name="SecurityMechanismParameterType">
    <xs:sequence>
      <xs:any/>
    </xs:sequence>
    <xs:attribute name="SecurityParameterId" type="xs:anyURI"/>
  </xs:complexType>
  <xs:element name="SecurityServices" type="wssqop:SecurityServicesType"/>
  <xs:complexType name="SecurityServicesType">
    <xs:sequence>
      <xs:element ref="wssqop:SecurityService" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="SecurityService" type="wssqop:SecurityServiceType"/>
  <xs:complexType name="SecurityServiceType">
    <xs:sequence>
      <xs:element ref="wssqop:SecurityServiceTargets"/>
      <xs:element ref="wssqop:SecurityMechanismRefs"/>
    </xs:sequence>
    <xs:attribute name="SecurityServiceId" type="xs:anyURI"/>
  </xs:complexType>
  <xs:element name="SecurityServiceTargets" type="wssqop:SecurityServiceTargetsType"/>
  <xs:complexType name="SecurityServiceTargetsType">
    <xs:sequence>
      <xs:element ref="wssqop:SecurityServiceTarget" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="SecurityServiceTarget" type="wssqop:SecurityServiceTargetType"/>
  <xs:complexType name="SecurityServiceTargetType">
    <xs:choice>
      <xs:element ref="wssqop:SecurityServiceTargetRef"/>
      <xs:element ref="wssqop:SecurityServiceTargetPath"/>
    </xs:choice>
    <xs:attribute name="SecurityServiceTargetId" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:element name="SecurityServiceTargetRef" type="xs:string"/>
  <xs:element name="SecurityServiceTargetPath" type="xs:anyURI"/>
  <xs:element name="SecurityMechanismRefs" type="wssqop:SecurityMechanismRefsType"/>
  <xs:complexType name="SecurityMechanismRefsType">
    <xs:sequence>
      <xs:element name="SecurityMechanismRef" type="xs:QName" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Appendix B – WSDL secure-soap binding (normative)

This appendix indicates how <SecurityServices> and <SecurityMechanisms> elements are included in a WSDL for a secured SOAP interface definition. Additions to the WSDL 1.1 SOAP binding are highlighted.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://schemas.xmlsoap.org/wsdl/secure-soap/"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:secure-soap="http://schemas.xmlsoap.org/wsdl/secure-soap/"
xmlns:wssqop="urn:oasis:names:tc:wssqop:1.0">
  <xs:import namespace="urn:oasis:names:tc:wssqop:1.0" schemaLocation="D:\My
Data\Standards\QoP\v09\QoPSpec v09.xsd"/>
  <element name="SecurityMechanisms" type="wssqop:SecurityMechanismsType"/>
  <element name="binding" type="secure-soap:bindingType"/>
  <complexType name="bindingType">
    <attribute name="transport" type="anyURI" use="optional"/>
    <attribute name="style" type="secure-soap:styleChoice" use="optional"/>
  </complexType>
  <simpleType name="styleChoice">
    <restriction base="string">
      <enumeration value="rpc"/>
      <enumeration value="document"/>
    </restriction>
  </simpleType>
  <element name="operation" type="secure-soap:operationType"/>
  <complexType name="operationType">
    <sequence>
      <element ref="secure-soap:SecurityServices"/>
    </sequence>
    <attribute name="soapAction" type="anyURI" use="optional"/>
    <attribute name="style" type="secure-soap:styleChoice" use="optional"/>
  </complexType>
  <element name="SecurityServices" type="wssqop:SecurityServicesType"/>
  <element name="body" type="secure-soap:bodyType"/>
  <complexType name="bodyType">
    <attribute name="encodingStyle" type="anyURI" use="optional"/>
    <attribute name="parts" type="NMTOKENS" use="optional"/>
    <attribute name="use" type="secure-soap:useChoice" use="optional"/>
    <attribute name="namespace" type="anyURI" use="optional"/>
  </complexType>
  <simpleType name="useChoice">
    <restriction base="string">
      <enumeration value="literal"/>
      <enumeration value="encoded"/>
    </restriction>
  </simpleType>
  <element name="fault" type="secure-soap:faultType"/>
  <complexType name="faultType">
    <complexContent>
      <restriction base="secure-soap:bodyType">
        <attribute name="parts" type="NMTOKENS" use="prohibited"/>
      </restriction>
    </complexContent>
  </complexType>
  <element name="header" type="secure-soap:headerType"/>
  <complexType name="headerType">
    <all>
      <element ref="secure-soap:headerfault"/>
    </all>
    <attribute name="message" type="QName" use="required"/>
    <attribute name="parts" type="NMTOKENS" use="required"/>
  </complexType>

```

```
<attribute name="use" type="secure-soap:useChoice" use="required"/>
<attribute name="encodingStyle" type="anyURI" use="optional"/>
<attribute name="namespace" type="anyURI" use="optional"/>
</complexType>
<element name="headerfault" type="secure-soap:headerfaultType"/>
<complexType name="headerfaultType">
  <attribute name="message" type="QName" use="required"/>
  <attribute name="parts" type="NMTOKENS" use="required"/>
  <attribute name="use" type="secure-soap:useChoice" use="required"/>
  <attribute name="encodingStyle" type="anyURI" use="optional"/>
  <attribute name="namespace" type="anyURI" use="optional"/>
</complexType>
<element name="address" type="secure-soap:addressType"/>
<complexType name="addressType">
  <attribute name="location" type="anyURI" use="required"/>
</complexType>
</schema>
```

Appendix C – WSS header schema (normative)

The consumer policy for a service response is conveyed to the provider in a WSS header. Below is the schema for the header contents. The message to which a policy conveyed in this manner applies is indicated in the `wssqp:SecurityServices/SecurityService/SecurityServiceTargets/SecurityServiceTarget` element of the header contents. This may be an output or fault message associated with the operation to which the `<ConsumerPolicy>` element is attached, or (in the case of a privacy policy) it may be a message in another interface offered by the service provider.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wssqp="urn:oasis:names:tc:wssqp:1.0"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="urn:oasis:names:tc:wssqp:1.0" schemaLocation="D:\My
Data\Standards\QoP\v09\QoPSpec v09.xsd"/>
  <xs:element name="ConsumerPolicy" type="ConsumerPolicyType"/>
  <xs:complexType name="ConsumerPolicyType">
    <xs:sequence>
      <xs:element ref="wssqp:SecurityMechanisms"/>
      <xs:element ref="wssqp:SecurityServices"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```