JAX-RS XML Security

JAX-RS: XML Security

Introduction

CXF 2.5.0 introduces an initial support for securing JAX-RS clients and endpoints with XML Signature and XML Encryption.
This is a work in progress and the enhancements will be applied regularly. Support for the alternative signature and encryption technologies will also be provided in due time.

Maven dependencies

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-rs-security-xml</artifactId>
  <version>2.5.2</version>
</dependency>
```

Backwards compatibility configuration note

From Apache CXF 3.1.0, the WS-Security based configuration tags used to configure XML Signature or Encryption ("ws-security-*") have been changed to just start with "security-". Apart from this they are exactly the same. Older "ws-security-" values continue to be accepted in CXF 3.1.0. To use any of the configuration examples in this page with an older version of CXF, simply add a "ws-" prefix to the configuration tag.

XML Signature

XML Signature defines 3 types of signatures: enveloped, enveloping and detached. All the three types are supported by CXF JAX-RS.

New Starting from CXF 2.5.2 it is also possible to add XML Signatures on the server side and get them validated on the client side.

Enveloped signatures

Payload:

```
<Book ID="4bd59819-7b78-47a5-bb61-cc08348e9d48">
   <id>126</id>
   <name>CXF</name>

   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#4bd59819-7b78-47a5-bb61-cc08348e9d48">
           <ds:Transforms>
             <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
             <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
           </ds:Transforms>
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <ds:DigestValue>eFduzs6Cg1/Wd6jagUmr8vRYxHY=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
<ds:SignatureValue>DLD+wU85G+Q+H/SNoMr1I7tOCAZAjd3lYE84sBGU5tuMtzbwxKOIgg10g2F1SUbpujy1CZZ9BPkQNA+gA1CH4
FE3uiBzp3DDSVv6o5l6Q76Ci0XI28ylO7O1OCY+q2nbP0WtERFWOn9f9nniVKbduz6YQHjv6cNLd8pf4+k2U3g=</ds:SignatureValue>

       <ds:KeyInfo>
         <ds:X509Data><ds:
X509Certificate>MIICGjCCAYOgAwIBAgIESVRgATANBgkqhkiG9w0BAQUFADAzMRMwEQYDVQQKEwphcGFjaGUub3JnMQwwCgYDVQQL
EwNlbmcxDjAMBgNVBAMTBWN4ZmNhMB4XDTcwMDEwMTAwMDAwMFoXDTM4MDExOTAzMTQwN1owMzETMBEGA1UEChMKYXBhY2hlLm9yZzEMMAoGA1UE
CxMDZW5nMQ4wDAYDVQ
QDEwVhbGljZTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAvu747/VShQ85f16DGSc4Ixh9PVpGguyEqrCsK8q9XHOYX9l9
/g5wEC6ZcR2FwfNsoaHcKNPjd5sSTzVt
BWmQjfBEfIqwTR7vuihOxyNTwEzVwIJzvo7p8/aYxk+VdBtQxq4UweIcf
/iFkUbM1cZ1oiXRQzciRBi+C1BQCQE0qzsCAwEAAaM7MDkwIQYDVR0SBBowGIIWTk9UX0ZPUl
9QUk9EVUNUSU9OX1VTRTAUBgNVHREEDTALgglsb2NhbGhvc3QwDQYJKoZI
hvcNAQEFBQADgYEAhLwkm+8psKt4gnbikGzV0TgpSWGcWxWKBi+z8tI2n6hFA5v1jVHHa4G9h3s0nxQ2TewzeR
/k7gmgV2sI483NgrYHmTmLKaDBWza2pAuZuDhQH8GAEh
JakFtKBP++EC9rNNpZnqqHxx3qb2tW25qRtBzDmK921gg9PMomMc7uqRQ=</ds:X509Certificate>
        </ds:X509Data>

        <ds:KeyValue>
          <ds:RSAKeyValue>
             <ds:Modulus>vu747/VShQ85f16DGSc4Ixh9PVpGguyEqrCsK8q9XHOYX9l9
/g5wEC6ZcR2FwfNsoaHcKNPjd5sSTzVtBWmQjfBEfIqwTR7vuihOxyNTw
EzVwIJzvo7p8/aYxk+VdBtQxq4UweIcf/iFkUbM1cZ1oiXRQzciRBi+C1BQCQE0qzs=</ds:Modulus>
             <ds:Exponent>AQAB</ds:Exponent>
          </ds:RSAKeyValue>
        </ds:KeyValue>
       </ds:KeyInfo>
     </ds:Signature>

</Book>
```

Note that the Book root element is signed including its name and id children, and a signature ds:Reference links to Book.

Server Configuration fragment:

```
<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.BookStore"/>
<bean id="xmlSigHandler" class="org.apache.cxf.rs.security.xml.XmlSigInHandler"/>
<bean id="xmlSigOutHandler" class="org.apache.cxf.rs.security.xml.XmlSigOutInterceptor"/>

<jaxrs:server address="/xmlsig">
    <jaxrs:serviceBeans>
      <ref bean="serviceBean"/>
    </jaxrs:serviceBeans>
    <!--
       Required for validating the in signature and removing it from the payload.
       It also persists the signature on the current Message which can be disabled.
    -->
    <jaxrs:providers>
      <ref bean="xmlSigHandler"/>
    </jaxrs:providers>
    <!--
       Required for adding a new signature to the outbound payload
    -->
    <jaxrs:outInterceptors>
          <ref bean="xmlSigOutHandler"/>
    </jaxrs:outInterceptors>

    <jaxrs:properties>
          <entry key="security.callback-handler"
                  value="org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback"/>
          <entry key="security.signature.properties"
                  value="org/apache/cxf/systest/jaxrs/security/alice.properties"/>
    </jaxrs:properties>
</jaxrs:server>
```

Note that org.apache.cxf.rs.security.xml.XmlSigInHandler is responsible for validating the signature attached to the inbound payload and is capable of processing all 3 types of XML Signature.

org.apache.cxf.rs.security.xml.XmlSigOutInterceptor is responsible for adding a new signature to the outbound payload.

Client code:

```
String address = "https://localhost:8080/xmlsig/bookstore/books";
JAXRSClientFactoryBean bean = new JAXRSClientFactoryBean();
bean.setAddress(address);

// setup properties
Map<String, Object> properties = new HashMap<String, Object>();
properties.put("security.callback-handler",
               "org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback");
properties.put("security.signature.username", "alice");
properties.put("security.signature.properties",
               "org/apache/cxf/systest/jaxrs/security/alice.properties");
bean.setProperties(properties);

// add the interceptor which will add a signature to the outbound payload
XmlSigOutInterceptor sigOutInterceptor = new XmlSigOutInterceptor();
bean.getOutInterceptors().add(sigOutInterceptor);

// add the interceptor which will validate a signature in the inbound payload
XmlSigInInterceptor sigInInterceptor = new XmlSigInInterceptor();
bean.getInInterceptors().add(sigInInterceptor);


// load a bus with HTTPS configuration:
SpringBusFactory bf = new SpringBusFactory();
Bus bus = bf.createBus(configLocation);
bean.setBus(bus);

// use WebClient (or proxy) as usual
WebClient wc = bean.createWebClient();
Book book = wc.post(new Book("CXF", 126L), Book.class);
```

Spring configuration can also be used.
Please also check Secure JAX-RS Services on how HTTPS can be configured from Spring.

Enveloping signatures

Payload:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#88e688e6-6512-406f-9e88-a58e5d781ff0">
        <ds:Transforms>
           <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>Cq3zl3t3DqWTvuZ+4EtZgGs4ikk=</ds:DigestValue>
      </ds:Reference>
   </ds:SignedInfo><ds:SignatureValue>NvcCS8vx3YJkc8fHMf8bQkC+lwasC6CwiS7HfKSm8t+6TtYdM7TRbYxSuqfCTkF4
vBIldWIzl6UngON592FfJdbvrgE2CusCkIybrP7BBmP7zTSV0GjH4/60L6ObkhGPkMNoKzw4V+zgF7Zo+F7ngsz5ZUWZX/GWETmTtYtcfT0=<
/ds:SignatureValue>
   <ds:KeyInfo>
     <ds:X509Data>
       <ds:X509Certificate><!-- Omitted for brevity--></ds:X509Certificate>
     </ds:X509Data>
     <ds:KeyValue>
      <ds:RSAKeyValue><ds:Modulus>vu747/VShQ85f16DGSc4Ixh9PVpGguyEqrCsK8q9XHOYX9l9/g5wEC6ZcR2FwfNsoaHcKNPjd5sST
zVtBWmQjfBEfIqwTR7vuihOxyNTwEzVwIJzvo7p8/aYxk+VdBtQxq4UweIcf/iFkUbM1cZ1oiXRQzciRBi+C1BQCQE0qzs=</ds:Modulus>
       <ds:Exponent>AQAB</ds:Exponent>
      </ds:RSAKeyValue>
     </ds:KeyValue>
   </ds:KeyInfo>
   <ds:Object ID="88e688e6-6512-406f-9e88-a58e5d781ff0">

      <Book>
         <id>126</id>
         <name>CXF</name>
      </Book>
   </ds:Object>
</ds:Signature>
```

This time the signature is enveloping the Book element using a ds:Object wrapper which ds:Reference links to.

Server Configuration fragment is identical to the one shown in the Enveloped signatures section.

Client code is nearly identical to the one shown in the Enveloped signatures section except that XmlSigOutInterceptor need to have an additional property set:

```
// add the interceptor dealing with adding a signature
XmlSigOutInterceptor sigInterceptor = new XmlSigOutInterceptor();
sigInterceptor.setStyle("enveloping");
```

Detached signatures

Payload:

```
<env:Envelope xmlns:env="http://org.apache.cxf/rs/env">

  <Book ID="e9836bc2-cb5a-453f-b967-a9ddbaf9a6de">
    <id>125</id>
    <name>CXF</name>
   </Book>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <ds:SignedInfo>
       <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
       <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
       <ds:Reference URI="#e9836bc2-cb5a-453f-b967-a9ddbaf9a6de">
         <ds:Transforms>
           <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         </ds:Transforms>
         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>Pxz77Hlg6I/MRsJz4gixkaMFtYI=</ds:DigestValue>
       </ds:Reference>
     </ds:SignedInfo>
<ds:SignatureValue>JSwgiVqZT1EtJ9xqtb90juS54pvZguzFMne7cQyGMQDvBW7b65aAAIfVx
/PmFB7Tuy4qB4zqNFCzCwHlhDurNP9NYB7PEzFsA3v
3vSyEcHnpUhu41xmBvjT5HWEKbuzqX0dHekizuUefbfzG5WpluVPmOgjashrm9DIhfEf+Hyg=</ds:SignatureValue>
     <ds:KeyInfo>
      <ds:X509Data>
         <ds:X509Certificate><!--Omitted for Brewity--></ds:X509Certificate>
      </ds:X509Data>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>vu747/VShQ85f16DGSc4Ixh9PVpGguyEqrCsK8q9XHOYX9l9
/g5wEC6ZcR2FwfNsoaHcKNPjd5sSTzVtBWmQjfBEfIqwTR7v
uihOxyNTwEzVwIJzvo7p8/aYxk+VdBtQxq4UweIcf/iFkUbM1cZ1oiXRQzciRBi+C1BQCQE0qzs=</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
     </ds:KeyInfo>
   </ds:Signature>

    <saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001
/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ID="_E462768C678896CE9913202742137181"
IssueInstant="2011-11-02T22:50:13.718Z" Version="2.0" xsi:type="saml2:AssertionType">

<saml2:Issuer>https://idp.example.org/SAML2</saml2:Issuer>

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
 <!--
    Enveloped/embedded SAML Assertion XML Signature is omitted for brevity
    See the JAX-RS SAML section for more info
 -->
</ds:Signature>
<!-- the rest of SAML assertion -->
</saml2:Assertion>
</env:Envelope>
```

Note that the whole payload is enveloped by a configurable element wrapper. The Book instance is one part of the envelope and it's signed by a detached signature (see the first ds:Signature, with its ds:Reference linking to Book). The envelope also has an embedded SAML assertion which has its own enveloped signature.

The instance of org.apache.cxf.rs.security.xml.XmlSigInHandler will handle a detached XML signature of the Book XML fragment on the server side. See the JAX-RS SAML for more info on how to deal with SAML assertions.

Client code is nearly identical to the one shown in the Enveloped signatures section except that XmlSigOutInterceptor need to have an additional property set:

```
// add the interceptor dealing with adding a signature
XmlSigOutInterceptor sigInterceptor = new XmlSigOutInterceptor();
sigInterceptor.setStyle("detached");
```

Customizing the signature

org.apache.cxf.rs.security.xml.XmlSigOutInterceptor manages the creation of the signature on the client side.
The following properties can be set on it at the moment:

"style": possible values are "enveloped" (default), "enveloping" and "detached"
"envelopedName": only used with the "detached" style, default is "{http://org.apache.cxf/rs/env}Envelope"
"signatureAlgorithm": default is "http://www.w3.org/2000/09/xmldsig#rsa-sha1"
"digestAlgorithm": default is "http://www.w3.org/2000/09/xmldsig#sha1"

Signature Key Info Validation

By default ds:Signature is expected to contain ds:KeyInfo element.

Setting a "keyInfoMustBeAvailable" property to false on the out interceptors will lead to KeyInfo not included.

If the same property is set to false on the in interceptors then either an authenticated Principal name or a default store alias will be used to load the certificate for validating the signature.

XML Encryption

Encrypting XML payloads makes it possible to drop a requirement for HTTPS.

Here is a payload example:

```
<xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:RetrievalMethod Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
    <xenc:EncryptedKey Id="EK-B353DDCEE7C575B6A213203188664772">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <ds:KeyInfo>
            <ds:X509Data>
               <ds:X509Certificate><!-- Omitted for brevity --></ds:X509Certificate>
           </ds:X509Data>
        </ds:KeyInfo>
        <xenc:CipherData><xenc:CipherValue>tPtZz4pnVWquaV2a7O0y+VrHoeWwk3Eu5Jnu3RHz5rGDB
/MLyG6rBamhit03J2xWaV52zUtDAPEj8sr4oy5y2KLB09Hu317IbQjinePabUpd
+DLnwNn5iHZpHWJPfndkh07JdYZSrMwqOvJ3fqrNJ+LQeLzZDneT8sC1vRyhSDU=</xenc:CipherValue>
        </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
     <xenc:CipherValue>3ZPQ3SapAxemJwqG58sWh+r8B5SMRf/DZ2w/REswgl0zr8kpk0x4tayC5hl7IbSE8CPQYYHX8sXVnUFUoHOtJA==<
/xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData>
```

Here is a server configuration fragment:

```
<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.BookStore"/>
<bean id="xmlSigInHandler" class="org.apache.cxf.rs.security.xml.XmlSigInHandler"/>
<bean id="xmlEncInHandler" class="org.apache.cxf.rs.security.xml.XmlEncInHandler"/>

<jaxrs:server address="/xmlsig">
    <jaxrs:serviceBeans>
      <ref bean="serviceBean"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
       <ref bean="xmlEncHandler"/>
       <ref bean="xmlSigHandler"/>
    </jaxrs:providers>
     <jaxrs:properties>
           <entry key="security.callback-handler"
                  value="org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback"/>
           <entry key="security.encryption.properties"
                  value="org/apache/cxf/systest/jaxrs/security/bob.properties"/>
           <entry key="security.signature.properties"
                  value="org/apache/cxf/systest/jaxrs/security/alice.properties"/>
     </jaxrs:properties>
</jaxrs:server>
```

This configuration supports receiving signed and then encrypted XML payloads.

The code:

```
String address = "https://localhost:8080/xmlencryption/bookstore/books";
JAXRSClientFactoryBean bean = new JAXRSClientFactoryBean();
bean.setAddress(address);

// setup properties
Map<String, Object> properties = new HashMap<String, Object>();

properties.put("security.callback-handler",
               "org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback");
properties.put("security.encryption.username", "bob");
properties.put("security.encryption.properties",
                      "org/apache/cxf/systest/jaxrs/security/bob.properties");

// if signature required:
properties.put("security.signature.username", "alice");
properties.put("security.signature.properties",
               "org/apache/cxf/systest/jaxrs/security/alice.properties");

bean.setProperties(properties);

// if signature required: add the interceptor dealing with adding a signature
XmlSigOutInterceptor sigInterceptor = new XmlSigOutInterceptor();
bean.getOutInterceptors().add(sigInterceptor);

// add the interceptor dealing with the encryption

XmlEncOutInterceptor encInterceptor = new XmlEncOutInterceptor();
encInterceptor.setSymmetricEncAlgorithm("http://www.w3.org/2001/04/xmlenc#aes128-cbc");
bean.getOutInterceptors().add(encInterceptor);


// use WebClient (or proxy) as usual
WebClient wc = bean.createWebClient();
Response r = wc.post(new Book("CXF", 126L), Book.class);
assertEquals(200, r.getStatus());
```

Note that XmlEncOutInterceptor interceptor has a "symmetricEncAlgorithm" property set to a weaker type just to get CXF tests passing.

The actual application client code does not expect a payload such as Book back but if it did then configuring the server to encrypt the response would be straightforward:

```
<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.BookStore"/>
<bean id="xmlSigInHandler" class="org.apache.cxf.rs.security.xml.XmlSigInHandler"/>
<bean id="xmlSigOutHandler" class="org.apache.cxf.rs.security.xml.XmlSigOutInterceptor"/>

<bean id="xmlEncInHandler" class="org.apache.cxf.rs.security.xml.XmlEncInHandler"/>
<bean id="xmlEncOutHandler" class="org.apache.cxf.rs.security.xml.XmlEncOutInterceptor">
        <property name="symmetricEncAlgorithm" value="aes128-cbc"/>
</bean>

<jaxrs:server address="/xmlsec">
    <jaxrs:serviceBeans>
      <ref bean="serviceBean"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
       <ref bean="xmlEncInHandler"/>
       <ref bean="xmlSigInHandler"/>
    </jaxrs:providers>
    <jaxrs:outInterceptors>
        <ref bean="xmlSigOutHandler"/>
        <ref bean="xmlEncOutHandler"/>
     </jaxrs:outInterceptors>
     <jaxrs:properties>
         <entry key="security.callback-handler"
                  value="org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback"/>
         <entry key="security.encryption.properties"
                  value="org/apache/cxf/systest/jaxrs/security/alice.properties"/>
         <entry key="security.signature.properties"
                  value="org/apache/cxf/systest/jaxrs/security/bob.properties"/>

    </jaxrs:properties>
</jaxrs:server>
```

Now the client code can be updated to expect an encrypted and signed Book back:

```
// Use the previous code fragment, add the in interceptors:
XmlEncInInterceptor encInInterceptor = new XmlEncInInterceptor();
bean.getInInterceptors().add(encInInterceptor);
XmlSigInInterceptor sigInInterceptor = new XmlSigInInterceptor();
bean.getInInterceptors().add(sigInInterceptor);
```

Using the request signature certificates for the encryption

From CXF 2.6.1 and 2.5.4:

When multiple clients are posting the encrypted and signed payloads, the following configuration will lead to the request signature certificates being utilized for encrypting the symmetric key used to encrypt the response:

```
<!-- server -->
<jaxrs:server>
<jaxrs:properties>
         <entry key="security.callback-handler"
                  value="org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback"/>
         <entry key="security.encryption.properties"
                  value="org/apache/cxf/systest/jaxrs/security/alice.properties"/>
         <entry key="security.encryption.username" value="useReqSigCert"/>
         <entry key="security.signature.properties"
                  value="org/apache/cxf/systest/jaxrs/security/bob.properties"/>

    </jaxrs:properties>
</jaxrs:server>
<jaxrs:client>
    <jaxrs:properties>
         <entry key="security.callback-handler"
                  value="org.apache.cxf.systest.jaxrs.security.saml.KeystorePasswordCallback"/>
         <entry key="security.encryption.properties"
                  value="org/apache/cxf/systest/jaxrs/security/bob.properties"/>
         <entry key="security.encryption.username" value="bob"/>
         <entry key="security.signature.properties"
                  value="org/apache/cxf/systest/jaxrs/security/alice.properties"/>
         <entry key="security.signature.username" value="alice"/>
    </jaxrs:properties>
</jaxrs:client>
```

The "security.encryption.username" server property is set to "useReqSigCert".

Note that the client configuration assumes Alice (with its alice.properties) represents a given client, Bob (with its bob.properties) - the receiver/server.

On the server side the encryption properties point to alice.properties and signature.properties to bob.properties. This is because the outbound signature needs to be done with the Bob's certificate and the encryption - with either the specific Alice's certificate or the certificate from the inbound signature. Note that the in encryption handler will check the signature properties first - this will ensure that the Bob's certificate used to encrypt the data on the client side can be validated, similarly for the in signature handler.

Customizing the encryption

org.apache.cxf.rs.security.xml.XmlEncOutInterceptor manages the encryption process.
The following properties can be set on it at the moment:
"symmetricEncAlgorithm": default is "http://www.w3.org/2001/04/xmlenc#aes256-cbc", complete URIs or short identifiers are supported, for example, "aes128-cbc" or "http://www.w3.org/2001/04/xmlenc#aes256-cbc".
"digestAlgorithm": optional, example "http://www.w3.org/2001/04/xmlenc#sha256" can be set.
"keyEncAlgorithm": default is "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"
"keyIdentifierType": default is "X509_KEY", "X509_ISSUER_SERIAL" is also supported - useful when the whole x509Certificate should not be embedded

GCM Algorithm and BouncyCastle provider

Please see Colm's blog for the information about the possible attack against XML Encryption and the GCM algorithm which needs to be used in order to prevent it.

Restricting encryption and signature algorithms

From CXF 2.6.1 and 2.5.4:

It is possible to configure the in encryption and signature handlers with the properties restricting the encryption and signature algorithms that clients can use, for example:

```
    <bean id="sigProps" class="org.apache.cxf.rs.security.xml.SignatureProperties">
       <property name="signatureAlgo"
                 value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
       <property name="signatureDigestAlgo"
                 value="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <property name="signatureC14Method"
                 value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
       <property name="signatureC14Transform"
                 value="http://www.w3.org/2001/10/xml-exc-c14n#"
/>
    </bean>

    <bean id="encProps" class="org.apache.cxf.rs.security.xml.EncryptionProperties">
       <property name="encryptionKeyTransportAlgo"
                 value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
       <property name="encryptionSymmetricKeyAlgo"
                 value="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
    </bean>

    <bean id="xmlSigInHandlerWithProps" class="org.apache.cxf.rs.security.xml.XmlSigInHandler">
        <property name="signatureProperties" ref="sigProps"/>
    </bean>

    <bean id="xmlEncInHandlerWithProps" class="org.apache.cxf.rs.security.xml.XmlEncInHandler">
        <property name="encryptionProperties" ref="encProps"/>
    </bean>

    <!-- the following ensures that the outbound handlers will use the same algorithms that the client used --
>
    <bean id="xmlSigOutHandlerWithProps" class="org.apache.cxf.rs.security.xml.XmlSigOutInterceptor">
        <property name="signatureProperties" ref="sigProps"/>
    </bean>

    <bean id="xmlEncOutHandlerWithProps" class="org.apache.cxf.rs.security.xml.XmlEncOutInterceptor">
        <property name="encryptionProperties" ref="encProps"/>
    </bean>
```

Getting the same SignatureProperties and EncryptionProperties beans (with "sigProps" and "encProps" ids) registered with the outbound handlers will ensure that the algorithms used by the current client have not only been validated on the inbound side but also used on the outbound side for encrypting and signing the data.

Note that from CXF 2.7.1, 2.6.4 and 2.5.7, the XmlEncInHandler will require that the RSA-OAEP algorithm be used as the key transport encryption algorithm by default. As this algorithm is used by default by the XmlEncOutInterceptor, no action is required unless you are specifying a different algorithm on the outbound side. In this case, an EncryptionProperties object will need to be configured on XmlEncInHandler with the desired key transport algorithm.

Interoperability

The payloads containing the enveloping XML Signatures are structured according to the XML Signature specification and as such can be consumed by any XML Signature aware consumers capable of handling the enveloping signatures and extracting the signed payload.

Same applies to enveloped signatures, for example, a signed SAML assertion always contains an enveloped signature.

The way CXF creates detached XML Signatures is experimental, so at the moment CXF will be required on both ends for the detached signatures be created and validated.

The current XML Encryption support is in line with the specification and thus the capable non-CXF consumers will be able to decrypt the payloads.