

# JAX-RS SAML

## JAX-RS: SAML

- [Introduction](#)
- [Backwards compatibility configuration note](#)
- [Maven dependencies](#)
- [Enveloped SAML assertions](#)
- [SAML assertions in Authorization header](#)
- [SAML assertions as Form values](#)
- [Creating SAML Assertions](#)
- [SAML Assertion Validation](#)
  - [Validating SAML Subjects](#)
- [SAML Authorization](#)
  - [Claims Based Access Control](#)
  - [Role Based Access Control](#)
- [SAML Web SSO Profile](#)

## Introduction

CXF 2.5.0 introduces an initial support for working with [SAML2](#) assertions. So far the main focus has been put on making sure SAML assertions can be included in HTTP requests targeted at application endpoints: embedded inside XML payloads or passed as encoded HTTP header or form values.

See also [JAX-RS XML Security](#).

## 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.

## Maven dependencies

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxfrt-rs-security-xml</artifactId>
  <version>2.5.0</version>
</dependency>
```

This module depends on Apache WSS4J, as it contains a lot of useful utility code based around OpenSAML.

## Enveloped SAML assertions

Payload:

```

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

<Book ID="67ca6441-0c4e-4430-af0e-9463ce9226aa">
  <id>125</id>
  <name>CXF</name>
</Book>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <!-- Book signature, omitted for brevity -->
</ds:Signature>

<!-- SAML assertion with an enveloped 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="_62D574706635C0B9F413203247720501" IssueInstant="2011-11-03T12:52:52.050Z" 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#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#_62D574706635C0B9F413203247720501">
        <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#" />
            <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="xs" />
          </ds:Transform>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>IDD9nFocVm/7FpUbiGI3ZvpY2ps=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>JA2I7u/SmNsXGgWNdrLSovkipiM3JmGHsmpoP0EeIowPwnLMx0WvV0C3xNGNiT1jOBe2uv8+WchtPoppGTC2JTJVX
/t8PmKQCYzo4kVJo6Nms...</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate><!-- Omitted for brevity --></ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
  </ds:Signature>

  <saml2:Subject>
    <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="www.mock-sts.com"
>uid=sts-client,o=mock-sts.com</saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches"/>
  </saml2:Subject>

  <saml2:Conditions NotBefore="2011-11-03T12:52:52.063Z" NotOnOrAfter="2011-11-03T12:52:52.063Z">
    <saml2:AudienceRestriction>
      <saml2:Audience>https://sp.example.com/SAML2</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AuthnStatement AuthnInstant="2011-11-03T12:52:51.981Z" SessionIndex="123456">
    <saml2:AuthnContext><saml2:AuthnContextClassRef/></saml2:AuthnContext>
  </saml2:AuthnStatement>

  <saml2:AttributeStatement>
    <saml2:Attribute FriendlyName="subject-role"
      Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
      NameFormat="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <saml2:AttributeValue xsi:type="xs:string">user</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute Name="http://claims/authentication"
      NameFormat="http://claims/authentication-format">
      <saml2:AttributeValue xsi:type="xs:string">password</saml2:AttributeValue>
    </saml2:Attribute>
  </saml2:AttributeStatement>
</saml2:Assertion>
</env:Envelope>

```

Note that Book and SAML assertion are individually signed but the envelope wrapper itself is not.

Here is another payload showing the whole enveloped signed including Book and SAML Assertion, this time only a single signature will be available:

```
<env:Envelope xmlns:env="http://org.apache.cxf/rs/env" ID="e795cdd1-c19d-4a5c-8d86-e8a781af4787">

<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ID="_C76E3D5BBEE4C4D87913203281641141" IssueInstant="2011-11-03T13:49:24.114Z" Version="2.0" xsi:type="saml2:AssertionType">
  <saml2:Issuer>https://idp.example.org/SAML2</saml2:Issuer>
  <saml2:Subject>
    <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" NameQualifier="www.mock-sts.com"
      >uid=sts-client,o=mock-sts.com</saml2:NameID>
    <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches"/>
  </saml2:Subject>
  <saml2:Conditions NotBefore="2011-11-03T13:49:24.127Z" NotOnOrAfter="2011-11-03T13:49:24.127Z">
    <saml2:AudienceRestriction>
      <saml2:Audience>https://sp.example.com/SAML2</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AuthnStatement AuthnInstant="2011-11-03T13:49:24.044Z" SessionIndex="123456">
    <saml2:AuthnContext>
      <saml2:AuthnContextClassRef/>
    </saml2:AuthnContext>
  </saml2:AuthnStatement>
  <saml2:AttributeStatement>
    <saml2:Attribute FriendlyName="subject-role" Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
      NameFormat="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">user</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute Name="http://claims/authentication" NameFormat="http://claims/authentication-format">
      <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">password</saml2:AttributeValue>
    </saml2:Attribute>
  </saml2:AttributeStatement>
</saml2:Assertion>

<Book>
  <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="#e795cdd1-c19d-4a5c-8d86-e8a781af4787">
      <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>GR1pHd2JpxYiCz16ouCmTZjq/AA=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>C2qUDOFwart2GHFjX6kB3E3z73AMXtRR/6Qjgyp6XP/vTn/Fr2epDNub3q+gNdT0KgjLE2rSynM3QTcpHov9C8...</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:X509Data><ds:X509Certificate><!-- Omitted for brevity --></ds:X509Certificate></ds:X509Data>
    <ds:KeyValue><ds:RSAKeyValue><ds:Modulus>vu747/VShQ85f16DGSc4Ixh9...</ds:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyValue>
  </ds:KeyInfo>
</ds:Signature>
</env:Envelope>
```

Server configuration fragment:

```

<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.BookStore"/>
<bean id="samlHandler" class="org.apache.cxf.rs.security.saml.SamlEnvelopedInHandler"/>

<!-- only needed if the detached signature signing the application data is expected -->
<bean id="xmlSigHandler" class="org.apache.cxf.rs.security.xml.XmlSigInHandler"/>

<jaxrs:server
    address="https://localhost:${testutil.ports.jaxrs-saml}/samlxml">
    <jaxrs:serviceBeans>
        <ref bean="serviceBean"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
        <ref bean="xmlSigHandler"/>
        <ref bean="samlHandler"/>
    </jaxrs:providers>
    <jaxrs:properties>
        <entry key="security.signature.properties"
            value="org/apache/cxf/systest/jaxrs/security/alice.properties"/>
    </jaxrs:properties>
</jaxrs:server>

```

Client code:

```

private WebClient createWebClient(String address) {
    JAXRSClientFactoryBean bean = new JAXRSClientFactoryBean();
    bean.setAddress(address);

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

    bean.getOutInterceptors().add(new SamlEnvelopedOutInterceptor(!selfSigned));
    XmlSigOutInterceptor xmlSig = new XmlSigOutInterceptor();
    if (selfSigned) {
        xmlSig.setStyle(XmlSigOutInterceptor.DETACHED_SIG);
    }
    return bean.createWebClient();
}

```

When we also need to sign the application payload such as Book we need to make sure that a detached XML signature for Book is created. When the whole envelope is signed then SamlEnvelopedOutInterceptor needs to be placed before XmlSigOutInterceptor hence the "new SamlEnvelopedOutInterceptor(!selfSigned)" constructor is invoked.

## SAML assertions in Authorization header

Logging output:

```

Address: https://localhost:9000/samlheader/bookstore/books/123
Http-Method: GET
Headers: {Accept=[application/xml], Authorization=[SAML
eJydVlmTokgQfu9fYTCPrs2htGKMHVEcKq2gKOlxsoFQAsqhFAjNr99CW1ud7t2ZjdA...], ...}

```

Note that the Authorization header has an encoded SAML Assertion as its value. The original SAML assertion has been optionally compressed using a deflated encoding and then base64-encoded. This encoded value can be signed itself - but it is not currently possible.

Server configuration is similar to the one from the Enveloped SAML Assertions section, the only difference is that a SAML handler needs to be replaced:

```

<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.BookStore"/>
<bean id="samlHandler" class="org.apache.cxf.rs.security.saml.SamlHeaderInHandler"/>

<!-- same as in the Enveloped SAML Assertions section -->

```

Client code:

```

private WebClient createWebClient(String address) {
    JAXRSClientFactoryBean bean = new JAXRSClientFactoryBean();
    bean.setAddress(address);

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

    bean.setProperties(properties);

    bean.getOutInterceptors().add(new SamlHeaderOutInterceptor());

    return bean.createWebClient();
}

```

## SAML assertions as Form values

Logging output:

```

Address: https://localhost:9000/samlform/bookstore/books
Encoding: ISO-8859-1
Http-Method: POST
Content-Type: application/x-www-form-urlencoded
Headers: {Accept=[application/xml], Cache-Control=[no-cache], connection=[keep-alive], Content-Length=[2206],
content-type=[application/x-www-form-urlencoded],
Host=[localhost:9000], Pragma=[no-cache], User-Agent=[Apache CXF ${project.version}]}
Payload: name=CXF&id=125&SAMLToken=eJydVltzqkgQfs+vsDiPwCNFjWIdUzUIGqJgQMQLyxYOI6BclAFBfv0OGo16kt1ztk...

```

Note that only form 'name' and 'id' fields will remain after the SAML handler processes a SAML assertion encoded in the SAMLToken form field. The original SAML assertion has been optionally compressed using a deflated encoding and then base64-encoded. This encoded value can be signed - but it is not currently possible.

Server configuration is similar to the one from the Enveloped SAML Assertions section, the only difference is that a SAML handler needs to be replaced:

```

<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.BookStore"/>
<bean id="samlHandler" class="org.apache.cxf.rs.security.saml.SamlFormInHandler"/>

<!-- same as in the Enveloped SAML Assertions section -->

```

The client code is the same as in the SAML assertions in Authorization header section except than an instance of SamlFormOutInterceptor has to be registered:

```

bean.getOutInterceptors().add(new SamlFormOutInterceptor());

```

## Creating SAML Assertions

If you use CXF JAX-RS client API to experiment with SAML then all you need to do is to register an appropriate out interceptor as shown in the above code fragments. The interceptor will ensure that a SAML assertion is created and added inside the XML envelope, as a form or HTTP header value. All of the SAML output interceptors depend on a "security.saml-callback-handler" property linking to a custom javax.security.auth.callback.Callback implementation which in its handle(Callbacks) method provides the information which is needed to create a SAML assertion to a org.apache.ws.security.saml.ext.SAMLCallback Callback instance, for example, see this [custom implementation](#).

More involved cases with SAML assertions being created by identity providers will be supported, with the help of CXF (WS) STSClient when needed.

## SAML Assertion Validation

When SAML assertions are received on the server side, they are validated to make sure that the enveloped signatures are correct. SubjectConfirmation methods (sender-vouches, holder-of-key, bearer) are also checked. The validation can be delegated to STS if needed. By default, server side SAML handlers have a "samlValidator" property set to an instance of org.apache.ws.security.validate.SamlAssertionValidator which does a thorough validation of the assertion. If needed org.apache.cxf.ws.security.trust.STSTokenValidator can be set instead which will use STS to validate the assertion. Custom validators extending WSS4J SamlAssertionValidator and doing the additional application-specific validation can be registered if needed.

Note the fact that the default validation relies a lot on the code heavily utilized by the WS-Security implementation should be of no concern - it is an example of the integration on its own in order to get the validation done. For example, WS-\* STS are heavily used in the enterprise today and it simply makes a complete sense to rely on it to validate a SAML assertion if it is possible.

SubjectConfirmation sender-vouches and holder-of-key methods can be easily validated with enveloped SAML assertions given that the embedded SAML signatures and key info can be checked against the signature used to sign the envelope or a custom payload like Book.

At the moment these methods can not be properly validated when the assertion is provided in a header or in the form, the additional signature signing the encoded SAML token will be needed - this will be supported in due time. Use "bearer" in those cases.

## Validating SAML Subjects

The first and most important thing which needs to be done is to verify that an assertion Subject can be mapped to a recognized identity instance.

There is a number of ways a Subject can be validated.

If STS is asked to validate the assertion then a successful response from IDP will likely be good enough for CXF to trust the identity of the provider. If the assertion signature is verified locally using the public key of IDP then it could a good enough confirmation too.

Alternatively, a custom validator, extending either [org.apache.ws.security.validate.SamlAssertionValidator](#) or CXF SAML [SecurityContextProvider implementation](#) can be registered with the server side SAML handler.

The latter option is preferred because not only one can validate Subject - but also ensure that a resulting SecurityContext will return a user Principal with a proper name - given that the actual Subject name available in the assertion may need to be translated to a name recognized by the local security stores or application. A combination of the assertion's Subject and AttributeStatement elements may need to be checked to establish a real name.

In cases like this you may want to register a custom SecurityContextProvider even if you have STS validating the assertion. Yet another reason is to retrieve the information about roles for a given Subject or map the assertion claims to roles for working with the RBAC to succeed, see the next section for more information.

Have a look please at this server configuration example:

```
<bean id="serviceBeanClaims" class="org.apache.cxf.systest.jaxrs.security.saml.SecureClaimBookStore"/>
<bean id="samlEnvHandler" class="org.apache.cxf.rs.security.saml.SamlEnvelopedInHandler">
    <property name="securityContextProvider">
        <bean class="org.apache.cxf.systest.jaxrs.security.saml.CustomSecurityContextProvider"/>
    </property>
</bean>

<jaxrs:server address="/saml-claims">
    <jaxrs:serviceBeans>
        <ref bean="serviceBeanClaims"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
        <ref bean="samlEnvHandler"/>
    </jaxrs:providers>
</jaxrs:server>
```

## SAML Authorization

SAML assertions may contain so-called claims which are represented by a sequence of SAML AttributeStatements containing one or more Attributes, for example:

```
<saml2:Assertion>
<!-- ... -->
<saml2:AttributeStatement>
    <saml2:Attribute NameFormat="http://schemas.xmlsoap.org/ws/2005/05/identity/claims"
                    Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role"
                    FriendlyName="subject-role">
        <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">user</saml2:
    AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute NameFormat="http://claims/authentication"
                    Name="http://claims/authentication-format">
        <saml2:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">password</saml2:
    AttributeValue>
    </saml2:Attribute>
</saml2:AttributeStatement>
<!-- ... -->
</saml2:Assertion>
```

An individual claim is scoped by NameFormat and Name attribute. NameFormat is similar to a namespace, while Name identifies what the value of this claim represents, for example, in the above fragment two claims are provided, one has a value "user" which represents a role of the assertion's Subject, another one has a value of "password" which identifies the way Subject authenticated itself, i.e, Subject provided its password (presumably to IDP).

Now, what is interesting is to see if it is possible to use these claims with Role-Based Access-Control (for example, with endpoints relying on @RolesAllowed annotations) as well as with the more complex authorization logic (for example, let this resource be invoked only if Subject used a password to get authenticated at IDP).

## Claims Based Access Control

CXF JAX-RS supports Claims Based Access Control (CBAC) based on [Claim](#) and [Claims](#) annotations using claims extracted from SAML Assertions. Please see the [JAX-RS Token Authorization](#) page for more information.

## Role Based Access Control

CXF JAX-RS also supports Role Based Access Control (RBAC) based on role claims extracted from SAML Assertions. Please see the [JAX-RS Token Authorization](#) page for more information.

## SAML Web SSO Profile

Please see [this page](#) for more information