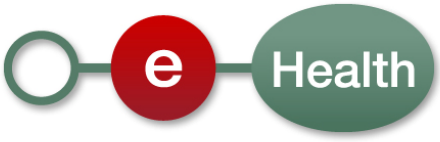


Cookbook
How to call a webservice
Version 0.02

The eHealth platform

Sint-Pietersteenweg 375

1040 BRUSSELS

1 Calling STS with JAXWS and Axis

Step 0: Requesting an eHealth Certificate and corresponding ETK using the ETEE requestor.

Step 1: Installation of *Axis2* and *Rampart* on your system:

- axis2-1.5.1 (http://apache.mogo.be//ws/axis2/1_5_1/axis2-1.5.1-bin.zip)
- rampart-1.4 (http://archive.apache.org/dist/ws/rampart/1_4/rampart-dist-1.4-bin.zip)

Step 2: Rampart comes with wss4j-1.5.4 which has a bug to use signed samlTokens.
You should replace this library with a higher version:

- wss4j-1.5.8 (http://apache.belnet.be/ws/wss4j/1_5_10/wss4j-bin-1.5.10.zip)

Step 3: The following rampart modules must be copied from rampart to axis configuration as explained in the axis documentation:

- rahas-1.4.mar
- rampart-1.4.mar

Step 4: Installation of the latest eHealth Crypto lib and his dependencies
https://www.ehealth.fgov.be/nl/page/website/home/platform/technicallibrary.html

- SIGNED-etee-crypto-1.X.X.jar
- dependencies of bouncycastle (bcmail / bcprov) (see readme of cryptolib)

Step 5: Copy default axis2.xml file to your project

Step 6: Edit local axis2.xml file (used to let axis know which SOAP security to configure)
Pieces in red must be updated to suit your configuration.

```
<axisconfig name="AxisJava2.0">
...
<module ref="rampart"/>
<parameter name="OutflowSecurity">
  <action>
   <items>Timestamp Signature</items>
   <user>alias of keypair in keystore defined in
         crypto.properties</user>
   <passwordCallbackClass>implementation of
         javax.security.auth.callback.CallbackHandler
         (to set password of keypair in keystore)
   </passwordCallbackClass>
   <signaturePropFile>crypto.properties</signaturePropFile>
   <signatureKeyIdentifier>DirectReference
   </signatureKeyIdentifier>
   <signatureParts>{Element}{http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd}Timestamp;Token;Body</signatureParts>
  </action>
</parameter>
...
</axisconfig>
```

Step 7: Create crypto.properties (used to get keystore of keypair to place signature on soap)

```
org.apache.ws.security.crypto.provider=
  org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=
  path.to.keyStore (must contain alias of user, defined in axis2.xml)
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=
  password.of.keyStore
```

Step 8: loadconfig()

```
import org.apache.axis2.deployment.FileSystemConfigurator;
import org.apache.axis2.jaxws.ClientConfigurationFactory;
import org.apache.axis2.metadata.registry.MetadataFactoryRegistry;

FileSystemConfigurator configurator = new
FileSystemConfigurator("path.to.axis.installion.folder.repo",
"path.to.local.axis.configfile");
ClientConfigurationFactory factory = new
ClientConfigurationFactory(configurator);
MetadataFactoryRegistry.setFactory(ClientConfigurationFactory.class,
factory);
```

Step 9: loading the correct certificates from keystore (p12

```
String p12location = "path.to.local.p12-file"  //see step 0
String p12password = "password.of.local.p12-file";

Security.addProvider(new BouncyCastleProvider());

X509Certificate inHOK = (X509Certificate)
KeyManager.getKeyStore(new File(p12location), "PKCS12",
password.toCharArray()).getCertificate("authentication");

X509Certificate inIdentification = (X509Certificate)
KeyManager.getKeyStore(new File(p12location), "PKCS12",
password.toCharArray()).getCertificate("authentication");

PrivateKey inIdentificationPK= (PrivateKey)
KeyManager.getKeyStore(new File(p12location), "PKCS12",
password.toCharArray()).getKey("authentication",
password.toCharArray());
```

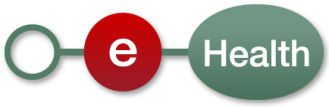
Step 10: build SAMLRequest: (see documentation opensaml)

```
SAMLAssertion assertion = new SAMLAssertion();

// adding assertion Attributes
assertion.setIssuer(inIndentification.getSubjectDN().toString());
assertion.setIssueInstant(Calendar.getInstance().getTime());

// adding samlConditions
assertion.setNotBefore(Calendar.getInstance().getTime());
Calendar notOnOrAfter = Calendar.getInstance();
notOnOrAfter.add(Calendar.SECOND, lifetime);
assertion.setNotOnOrAfter(notOnOrAfter.getTime());

//creating SAMLNameIdentifier
SAMLNameIdentifier inSAMLNameIdentifier = new
SAMLNameIdentifier(inIdentification.getSubjectDN().toString(),
inIdentification.getIssuerDN().toString(),
SAMLNameIdentifier.FORMAT_X509);
// adding attributeStatements
SAMLAttributeStatement attributeStatement = new
SAMLAttributeStatement();
assertion.addStatement(attributeStatement);
SAMLSubject attrStatementSubject = new SAMLSubject();
attrStatementSubject.setNameIdentifier(inSAMLNameIdentifier);
attributeStatement.setSubject(attrStatementSubject);

// adding every identification attribute
SAMLAttribute attribute_X = new SAMLAttribute();
attribute_X.setNamespace(...);
attribute_X.setName(...);
attribute_X.addValue(...);
attributeStatement.addAttribute(attribute_X);

SAMLSubject inSubject = new SAMLSubject();
```

```
inSubject.setConfirmationMethods(Collections.singletonList(
  SAMLSubject.CONF_HOLDER_KEY));

//creating SAMLNameIdentifier
SAMLNameIdentifier inSAMLNameIdentifier_2 = new
SAMLNameIdentifier(inIdentification.getSubjectDN().toString(),
inIdentification.getIssuerDN().toString(),
SAMLNameIdentifier.FORMAT_X509);
inSubject.setNameIdentifier(inSAMLNameIdentifier_2);

//adding HOK-certification information
Document scdDoc =
DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
Element subjectConfirmationData =
scdDoc.createElementNS("urn:oasis:names:tc:SAML:1.0:assertion",
"SubjectConfirmationData");
subjectConfirmationData.appendChild(assertion.toDOM(scdDoc));
inSubject.setConfirmationData(subjectConfirmationData);

Document keyInfoDoc =
DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
Element keyInfo =
keyInfoDoc.createElementNS("http://www.w3.org/2000/09/xmldsig#",
"ds:KeyInfo");
keyInfo.setAttributeNS("http://www.w3.org/2000/xmlns/", "xmlns:ds",
"http://www.w3.org/2000/09/xmldsig#");
Element x509Data =
keyInfoDoc.createElementNS("http://www.w3.org/2000/09/xmldsig#",
"ds:X509Data");
keyInfo.appendChild(x509Data);
Element x509Certificate =
keyInfoDoc.createElementNS("http://www.w3.org/2000/09/xmldsig#",
"ds:X509Certificate");
x509Data.appendChild(x509Certificate);
x509Certificate.setTextContent(new
String(Base64.encode(inHOK.getEncoded())));
inSubject.setKeyInfo(keyInfo);

// adding every requested Attributes
List<SAMLAttributeDesignator> attributeDesignators = new
ArrayList<SAMLAttributeDesignator>();

SAMLAttributeDesignator attributeDesignator_X = new
SAMLAttributeDesignator();
attributeDesignator_X.setNamespace(...);
attributeDesignator_X.setName(...);
attributeDesignators.add(attributeDesignator_X);

SAMLAttributeQuery inQuery = new SAMLAttributeQuery();
inQuery.setSubject(inSubject);
inQuery.setDesignators(attributeDesignators);

SAMLRequest inRequest = new SAMLRequest(inQuery);
```

Step 11: sign SAMLRequest (load keypaire and use SAMLRequest.sign()

```
inRequest.sign("http://www.w3.org/2000/09/xmldsig#rsa-sha1",
inIdentificationPK, Arrays.asList(inIdentification));
```



Step 12: send samlRequest

```
localWsdURL = see attachment
QName serviceQName = new QName("urn:be:fgov:ehealth:sts:protocol:v1",
                               "SecureTokenService");
QName portQName = new QName("urn:be:fgov:ehealth:sts:protocol:v1",
                            "SecureTokenServicePort");
String endpointUrl =
https://wwwacc.ehealth.fgov.be/sts_1_1/SecureTokenService
```

```
public SAMLResponse requestSecureToken(SAMLRequest samlRequest) throws
Exception {
// transform org.opensaml.SAMLRequest to source object
InputStream in = new
ByteArrayInputStream(samlRequest.toString().getBytes("UTF8"));


// create dispatch with local wsdl
Service service = Service.create(localWsdlURL, serviceQName);
Dispatch<Source> dispatch = service.createDispatch(portQName, Source.class,
Mode.PAYLOAD);

// set real endpoint
dispatch.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
endpointUrl);

Source response = dispatch.invoke(new StreamSource(in)); // dispatch to
webservice using jaxws

// transform source resultobject to org.opensaml.SAMLResponse
Transformer transformer =
TransformerFactory.newInstance().newTransformer();
DOMResult result = new DOMResult();
transformer.transform(response, result);
return new SAMLResponse(((Document)
result.getNode()).getDocumentElement());
}
```

2 Calling a JAXWS webservice protected with a SAML Token using Axis

Step 1: Installation of *Axis2* and *Rampart* on your system:

- axis2-1.5.1 (http://apache.mogo.be//ws/axis2/1_5_1/axis2-1.5.1-bin.zip)
- rampart-1.4 (http://archive.apache.org/dist/ws/rampart/1_4/rampart-dist-1.4-bin.zip)

Step 2: Rampart comes with wss4j-1.5.4 which has a bug to use signed samlTokens.
You should replace this library with a higher version:

- wss4j-1.5.8 (http://apache.belnet.be/ws/wss4j/1_5_10/wss4j-bin-1.5.10.zip)

Step 3: Te following rampart modules must be copied from rampart to axis configuration as explained in the axis documentation:

- rahas-1.4.mar
- rampart-1.4.mar

Step 4: Copy default axis2.xml file to your project

Step 5: Edit local axis2.xml file (used to let axis know which SOAP security to configure)
Pieces in red must be updated to suit your configuration.

```
<axisconfig name="AxisJava2.0">
...
<module ref="rampart" />
<parameter name="OutflowSecurity">
<action>
<items>Timestamp SAMLTokenSigned</items>
<user>alias of keypair in keystore (see further)</user>
<passwordCallbackClass>implementation of
javax.security.auth.callback.CallbackHandler</passwordCallbackClass>
<samlPropFile>saml.properties</samlPropFile>
<signaturePropFile>crypto.properties</signaturePropFile>
<signatureKeyIdentifier>X509KeyIdentifier</signatureKeyIdentifier>
<signatureParts>{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd}Timestamp;Body;</signatureParts>
</action>
</parameter>
...
</axisconfig>
```

Step 6: Implement method handle(..) of your CallBackHandler (used to get password of keypair)

```
/**
 *
 * @see  javax.security.auth.callback.CallbackHandler#handle(
        javax.security.auth.callback.Callback[])
 */

public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
 for (int i = 0; i < callbacks.length; i++) {
   WSPasswordCallback pwcb = (WSPasswordCallback) callbacks[i];
   pwcb.setPassword("get and set password of keypair in keystore");
 }
}
```

Step 7 : Create saml.properties and place under root of classpath (used to get samlToken)

```
org.apache.ws.security.saml.issuerClass=implementation of
org.apache.ws.security.saml.SAMLIssuer (constructor will take
samlPropFile as parameter)
# add properties to which you want access in your issuerClass
implementation
```

Step 8 : Implement method newAssertion() of your issuerClass
This should return a valid org.opensaml.SAMLAssertion stored on your system, database, … or get a new one from the eHealth STS.

Step 9: Create crypto.properties (used to get keystore of keypair to place signature on soap)

```
org.apache.ws.security.crypto.provider=
  org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=
  path.to.keyStore (must contain alias of user, defined in axis2.xml)
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=
  password.of.keyStore
```

Step 10: Add business code to your client

10.1 Installation of the axis clientconfig

```
import org.apache.axis2.deployment.FileSystemConfigurator;
import org.apache.axis2.jaxws.ClientConfigurationFactory;
import org.apache.axis2.metadata.registry.MetadataFactoryRegistry;

FileSystemConfigurator configurator = new
FileSystemConfigurator("path.to.axis.installion.folder.repo",
"path.to.local.axis.configfile");
ClientConfigurationFactory factory = new
ClientConfigurationFactory(configurator);
MetadataFactoryRegistry.setFactory(ClientConfigurationFactory.class,
factory);
```

10.2 create JAXWS service dispatcher

```
Service service = Service.create(localWsdlURL, serviceQName);
Dispatch<Source> dispatch = service.createDispatch(portQName,
Source.class, Mode.PAYLOAD);
```

10.3 set url of webservice

```
dispatch.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPER
TY, endpointUrl);
```

10.4 send message

```
Source response = dispatch.invoke(new StreamSource(new
StringReader(inputXml)));
```