SICS lightweight SAML library

Ludwig Seitz
SICS

September 30, 2011

Abstract

The SICS lightweight SAML library (SlSl) is a Java code library for creating and consuming SAML attribute assertions. This is a short documentation of that library

1 Introduction

The SAML [1] standard has good OpenSource support. Very comprehensive implementations of the standard exist. However with comprehensiveness comes a certain amount of overhead.

Therefore we have decided to provide a lightweight library targeted at a subset of SAML, namely attribute assertions.

2 Supported elements

The SlSl supports only the *assertion* part of the SAML standard. No support for *protocols, bindings, or profiles* is given, with the exception of the SAML profile of XACML [2], namely the part about converting XACML [3] attributes to SAML and vice-versa.

From the SAML assertions, the following elements are supported:

- NameID;
- Issuer;
- Assertion;
- Subject;
- SubjectConfirmation;
- SubjectConfirmationData;
- Signature (from the xmldsig standard);
- Conditions (with NotBefore and NotOnOrAfter attributes);
- OneTimeUse condition;

- AttributeStatement;
- Attribute;
- AttributeValue.

We do not support the following assertion elements:

- EnryptedID;
- AssertionIDRef;
- AssertionURIRef;
- EncryptedAssertion;
- AudienceRestriction condition;
- ProxyRestricting condition;
- Advice and all sub-elements;
- AuthnStatement and all sub-elements;
- EnryptedAttribute;
- AuthzDecisionStatement and all sub-elements.

3 Using the library to create assertions

In order to create an assertion (`SignedSAMLAssertion`) you need to create all the components of the assertion, except for the signature, first. These are:

- the issuer (`SAMLNameID`);
- the subject (`SAMLSubject`);
- the conditions (`SAMLConditions`);
- the statements (a `List` of `SAMLStatements`);

Furthermore you need to instantiate a `SAMLSigner` that will perform the signing of the assertion.

3.1 SAMLNameID

The `SAMLNameID` class takes one mandatory and four optional parameters.

- The `name` is a String value and is mandatory;
- the `qualifierType` is Boolean and a value of `true` means that we are providing a NameQualifier, while `false` means we are providing a SPNameQualifier (see section 2.2.2 of the SAML standard [1] for what this means);
- the `qualifier` is the String value of the NameQualifier or SPNameQualifier;

- the `format` is a `java.net.URI` representing the format of the NameID as defined in section 2.2.2 of the SAML standard [1];
- the `spProvidedID` is a name identifier established by a service provider, as defined in section 2.2.2 of the SAML standard [1].

3.2 SAMLSubject

Instances of the `SAMLSubject` class require either one of a `SAMLID` or a `List` of `SAMLSubjectConfirmation`.

Currently the only instance of the `SAMLID` interface is the `SAMLNameID` class (see section 3.1).

3.2.1 SAMLSubjectConfimation

The `SAMLSubjectConfirmation` class supports 3 types of standardized subject confirmation methods:

- urn:oasis:names:tc:SAML:2.0:cm:holder-of-key;
- urn:oasis:names:tc:SAML:2.0:cm:sender-vouches
- urn:oasis:names:tc:SAML:2.0:cm:bearer

These identifiers are available as the static variables `holder, sender` and `bearer` of the class.

The object constructor requires a `java.net.URI` method, a `List` of `SAMLID` representing the subject identifiers that are confirmed, and a `List` of `SAMLSubjectConfirmationData`.

The two latter parameters can both be null (although that's quite useless for confirming a subject).

3.2.2 SAMLSubjectConfirmationData

This class has two constructors, one for the holder-of-key and one for the two other methods of confirmation.

The constructors both take the following arguments, all of which are optional and can be null:

- notBefore, a `java.util.Date` before which the confirmation should not be used.
- notOnOrAfter, a `java.util.Date` on or after which the confirmation should not be used.
- recipient, a `java.net.URI` describing the recipient.
- inResponseTo, a String giving the SAML protocol message id to which this responds.
- address, a String describing the network address/location form which the attesting entity can present the assertion.

In addition the holder-of-key constructor also takes a `List` of `javax.xml.crypto.dsig.keyinfo.KeyInfo`, representing the key or keys that have been used to confirm the subject's identity.

3.3 SAMLConditions

The `SAMLConditions` class constructor takes 3 parameters, all of which are optional and can be null:

- notBefore, a `java.util.Date` before which the assertion should not be used.
- notOnOrAfter, a `java.util.Date` on or after which the assertion should not be used.
- a `List` of `SAMLCondition`.

3.3.1 SAMLCondition and SAMLOneTimeUse

The `SAMLCondition` class is abstract and is only instantiated by the `SAMLOneTimeUse` class. The latter class doesn't need any parameters and represents the fact that the assertion containing this condition should only be used once.

3.4 SAMLStatement and SAMLAttributeStatement

The `SAMLStatement` interface is currently only implemented by the `SAMLAttributeStatement` class.

The latter classes' constructor requires a `List` of `SAMLAttribute`.

3.4.1 SAMLAttribute

The `SAMLAttribute` class constructor requires a String describing the name of the attribute. When converting XACML attribute to SAML attributes, this is where the XACML AttributeId goes.

The other parameters of the constructor are optional and can be null.

- nameFormat is a `java.net.URI` describing the format of the attribute's name. For XACML attributes use the static variable `xamclNameFormat` of this class;
- friendlyName is a String giving a human-readable form of the name;
- otherXMLAttrs is a `Map` of Strings to Strings. This allows to specify other XML-attributes that should be included in the object.

  For XACML specify a mapping from the XACML DataType XML-attribute to it's value. If this mapping is missing, the attribute is assumed to have the DataType `http://www.w3.org/2001/XMLSchema#string`.

  If the XACML attribute has an Issuer XML-attribute this should be transferred to the Issuer element of the SAML assertion (see section 3).
- attributeValues is a `List` of `SAMLAttributeValue`/

3.4.2 SAMLAttributeValue

SAML attribute values take a String value and an optional `java.net.URI` type in the constructor. The type can be null.

3.5 SAMLSigner

The `SAMLSigner` class is used to generate signatures for `SignedSAMLAssertion`.

It requires the following parameters in its constructor:

- parser, an instance of `XMLInputParser`;
- privateKey, an instance of `java.security.PrivateKey`;
- cert, an instance of `java.security.cert.X509Certificate`.

3.5.1 XMLInputParser

The `XMLInputParser` class implements XML parsing. It's a convenience class for saving the lines of code needed to configure a `javax.xml.parsers.DocumentBuilder`.

It takes two parameters both of which can be null.

- schemas, an array of `java.io.InputStream` containing XML-schemas to use when parsing an XML document.
- entityMap, an instance of `Map` mapping String to String. used to resolve entities to local files. The keys are systemIds of publicIds of the entities and the values names of local files containing these entities.

3.6 Example code

```
SAMLSigner signer =

SAMLNameID subjectId = new SAMLNameID("ludwig@sics.se");
SAMLNameID issuer = new SAMLNameID("MSNP");

ArrayList<SAMLAttributeValue> values
        = new ArrayList<SAMLAttributeValue>();
values.add(new SAMLAttributeValue(null, "member"));
SAMLAttribute member = new SAMLAttribute(
                "group:SWiN", null, null, null, values);
List<SAMLAttribute> attrs = new ArrayList<SAMLAttribute>();
attrs.add(member);
SAMLAttributeStatement statement
        = new SAMLAttributeStatement(attrs);

List<SAMLStatement> statements = new ArrayList<SAMLStatement>();
statements.add(statement);

Date now = new Date();
Calendar cal = Calendar.getInstance();
cal.add(Calendar.DAY_OF_MONTH, 1);

Date notOnOrAfter = cal.getTime();
List<SAMLCondition> conditions = Collections.emptyList();
```

```
SAMLConditions conditionsE
        = new SAMLConditions(now, notOnOrAfter, conditions);

SignedSAMLAssertion assertion = null;
try {
        assertion = new SignedSAMLAssertion(issuer, subject,
        conditionsE, statements, signer);
    } catch (Exception e) {
        //FIXME: Do some error handling
    }
```

4 Using the library to consume assertions

In order to consume a SAMLAssertion you need to parse it into a `SignedSAMLAssertion` object. If the assertion has a signature, it will be verified with the certificate included in the signature.

The `SignedSAMLAssertion` class has a static method `getInstance` that will consume a `org.w3c.dom.Node` containing the SAML assertion.

This method will also perform signature verification and throw a `javax.xml.crypto.dsig.XMLSignatureException` if the signature is not valid.

4.1 Example code

```
Node samlAssertion = ...;
SignedSAMLAssertion newAssertion = null;
try {
        newAssertion
            = SignedSAMLAssertion.getInstance(samlAssertion);
} catch (VerificationException ve) {
        // FIXME: do something
} catch (MarshalException me) {
        // FIXME: do something
} catch (XMLSignatureException xse) {
        // FIXME: do something
}
```

References

[1] S. Cantor., J. Kemp., R. Philpott., E. Maler., eds.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0. Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2005) http://www.oasis-open.org.

[2] E. Rissanen., H. Lockhart., eds.: SAML 2.0 Profile of XACML Version 2.0. Committee specification, Organization for the Advancement of Structured Information Standards (OASIS) (2010) http://www.oasis-open.org/committees/xacml.

[3] S. Godik., T. Moses., eds.: eXtensible Access Control Markup Language (XACML). Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2003) http://www.oasis-open.org/committees/xacml.