Creating XML Signatures with Smart Cards

Technical Article 28 March 2007

Introduction

More and more cryptographic applications use smart cards to secure keys. Saving these keys and performing the cryptographic operation directly on the card protects the keys all time. With the PKCS#11-Provider Add-On, it's easy to develop applications that integrate smart cards. This article shows how to create XML signatures with XSECT using a smart card.

In order to adapt a typical XML-Signature application only two changes are needed. First, the application has to select the signing key from the PKCS#11 key store. Second, the XSECT provider has to delegate the private-key operations to the PKCS#11 provider, which performs the actual signing on the smart card.

Creation of the Signature

In this section, we demonstrate how to create an XML signature from an XML file and save it in a separate file.

The steps for creating an XML signature with a smart card are:

1. Registration of the required providers
2. Configuration of the delegation-provider in XSECT
3. Selection of the signature key from the PKCS#11 key store
4. Creation of an XML signature as usual

The initial step is to register the needed providers. The providers required for this demo are the IAIK-JCE provider for basic cryptographic operations, the XSECT provider for XML signatures and the PKCS#11 provider for the smart card integration. Before adding the XSECT provider, we configure it to delegate the signing operation to the PKCS#11 provider.

```
Security.addProvider(new IAIK());
pkcs11Provider_ = new IAIKPkcs11();
Security.addProvider(pkcs11Provider_);
XSecProvider xsecProvider = new XSecProvider();
XSecProvider.setDelegationProvider("Signature.SHA1withRSA",
      pkcs11Provider.getName());
Security.addProvider(xsecProvider);
```

Next, we select the signing key from the PKCS#11 key store. Note that the private key objects that we get from this key store do not contain the actual key material. They are only proxy objects. The key material never leaves the card.

```
KeyStore tokenKeyStore =
            pkcs11Provider_.getTokenManager().getKeyStore();
signatureKey_ = (PrivateKey) tokenKeyStore.getKey(keyAlias, null);
Certificate[] certificateChain =
            tokenKeyStore.getCertificateChain(keyAlias);
signingCertificate = (X509Certificate) certificateChain[0];
```

Now we can create an XML signature as usual with XSECT. In our example, we create a simple detached signature that contains the signing certificate in the `KeyInfo` element.

```
XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");
Reference ref = fac.newReference(
            dataURL, fac.newDigestMethod(DigestMethod.SHA1, null));

CanonicalizationMethod canonicalizationMethod =
            fac.newCanonicalizationMethod(
            CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
            (C14NMethodParameterSpec) null);
SignatureMethod signatureMethod = fac.newSignatureMethod(
            SignatureMethod.RSA_SHA1, null);
SignedInfo si = fac.newSignedInfo(
            canonicalizationMethod, signatureMethod,
            Collections.nCopies(1, ref));

KeyInfoFactory kif = fac.getKeyInfoFactory();
X509Data x509data = kif.newX509Data(
            Collections.nCopies(1, signingCertificate));
KeyInfo ki = kif.newKeyInfo(Collections.nCopies(1, x509data));

XMLSignature signature = fac.newXMLSignature(si, ki);

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
doc = dbf.newDocumentBuilder().newDocument();

DOMSignContext signContext = new DOMSignContext(
            (Key) signatureKey_, doc_);
signature.sign(signContext);
```

With the signature method `SignatureMethod.RSA_SHA1`, XSECT knows that it has to create a RSA signature with SHA-1 hashing. With the delegation feature, we specified that it should use the PKCS#11 provider if it wants to create a signature with the JCA algorithm name `SHA1withRSA`. The same mechanism works for other signature algorithms similarly.

JCA algorithm name	IAIK XSECT algorithm name
Signature.SHA1withRSA	SignatureMethod.RSA_SHA1
Signature.SHA256withRSA	XmldsigMore.SIGNATURE_RSA_SHA256
Signature.SHA1withECDSA	XmldsigMore.SIGNATURE_ECDSA_SHA1

Validation of the Signature

To validate the created XML signature no access to the smart card is needed, as the certificate was added to the signature. Therefore, the signature can be validated the same way as other XML signatures. A demo for the validation process can be downloaded below.

Summary

This article shows how to combine XML-Signature creation with XSECT and smart-card access with the PKCS#11 Provider. The example includes the signing certificate in the detached XML signature.

References

1. IAIK JCE Documentation:
   http://javadoc.iaik.tugraz.at/iaik_jce/current/index.html
2. IAIK XSECT Add-On Documentation:
   http://javadoc.iaik.tugraz.at/xsect/current/apidocs/xsect/index.html
3. IAIK PKCS#11-Provider Documentation:
   http://javadoc.iaik.tugraz.at/pkcs11_provider/current/index.html
4. Java SE 1.5.0 Documentation:
   http://java.sun.com/j2se/1.5.0/docs/api/
5. Signature algorithm names:
   http://java.sun.com/javase/6/docs/technotes/guides/security/StandardNames.html#Signature
   http://javadoc.iaik.tugraz.at/xsect/current/apidocs/xsect/iaik/xml/crypto/XmldsigMore.html