java_faq

Frequently Asked Questions - Apache XML Security for Java

1. Questions about Java

1.1. I have a Java-(security/cryptography) problem. Can you help me?

Go to the java forum of Oracle. You can find forums where you can ask questions like "How do I generate a keypair", etc.

1.2. I have a Java-XML problem.

Go to the java forum of Sun, section Java Technology & XML and have a look at Apache Xerces.

2. Questions about this package

2.1. How do I enable/turn off logging?

Apache Santuario XML Security for Java uses Apache Commons Logging. For more information see here.

2.2. What is the meaning of BaseURI?

When you work with URIs like "http://www.example.com/index.html", it is quite sure what you mean as this is an absolute URL, i.e. it is clear which protocol ise used to fetch which file from which server. When you use such a URL inside a signature, the software can automatically figure out what you sign. But when you sign something in your local file system or if you use a relative path like "../1.txt", it's not possible to understand this reference without some context. This context is the BaseURI. For instance, if you sign URI="../1.txt" and the BaseURI="file:///home/user/work/signature.xml", it is clear that the file BaseURI="file:///home/user/1.txt" is to be signed. But when you create the signature, the file BaseURI="file:///home/user/work/signature.xml" does not yet exist; therefore, you have to supply the URL where you intend to store the signature later (relative to the signed objects).

The String BaseURI is the systemID on which the Object will be stored in the future. This is needed to resolve relative links in the Reference elements which point to the filesystem or something similar.

Example: Imagine that you want to create a signature to store it on a web server as http://www.acme.com/signatures/sig1.xml. So BaseURI="http://www.acme.com/sig1.xml". This means that if you create a Reference with URI="./index.html", the library can easily use it's HTTPResourceResolver to fetch http://www.acme.com/index.html without that you have to say URI="http://www.acme.com/index.html".

2.3. How do I use the package to generate and verify a signature?

Checkout the samples here.

The samples divide into two groups: Samples that create and samples that verify Signatures. Eventually, you should adjust the verifying program to another filename if you get FileNotFoundExceptions.

2.4. I get a NullPointerException, and I don't know what's wrong.

Often, this problem is caused by using DOM1 calls like createElement(), setAttribute(), createAttribute(). These are non-namespace-aware and will cause XPath and C14N errors. Always use the DOM2 create(Attribute|Element)NS(...) methods instead, even if you're creating an element without a namespace (in that case, you can use null as a namespace).

The Xalan-J Team told us that DOM1 calls are deprecated and are not to be used in code. xml-security has been reviewed and is DOM1 clean now. The Xalan folks told us that if you create Elements or attributes using DOM1 calls which are not namespace aware, they do not care about any problem you have because of incorrect hehaviour of Xalan.

2.5. I sign a document and when I try to verify using the same key, it fails

After you have created the XMLSignature object, before you sign the document, you must embed the signature element in the owning document (using a call to XMLSignature.getElement() to retrieve the newly created Element node from the signature) before calling the XMLSignature.sign() method,

During canonicalisation of the SignedInfo element, the library looks at the parent and ancestor nodes of the Signature element to find any namespaces that the SignedInfo node has inherited. Any that are found are embedded in the canonical form of the SignedInfo. (This is not true when Exclusive Canonicalisation is used, but it is still good practice to insert the element node prior to the sign() method being called).

If you have not embedded the signature node in the document, it will not have any parent or ancestor nodes, so it will not inherit their namespaces. If you then embed it in the document and call verify(), the namespaces will be found and the canonical form of SignedInfo will be different to that generated during sign().

3. Resolver-Mania

3.1. Why do we need all these resolvers?

For security and comfort reasons. In the XML Security package, there exist many kinds of Resolvers for different purposes. Resolvers in this package do the same job as an EntityResolver in the SAX package: retrieve information from the apropriate location and give it to the parser/software who needs it. The reason for offering these different Resolvers is that it should be under complete control of the application which connections to the network are made. In the security area, it wouldn't be a good idea to imediately fetch some documents from the web or make other connections only because you want to verify a Signature. This resolver framework gives the application developer the ability to have total control about the interface from the library to the rest of the world.

3.2. ResourceResolvers

A ResourceResolver is used by a Reference to retrieve the signed resource from it's location. Different resolvers exist to get signed portions from the XML document in which the signature resides, to make HTTP connections or to fetch files from the local file system.

The concept of a ResourceResolver is very similar to an org.xml.sax.EntityResolver, but in contrast to that Interface, the ResourceResolver is able to de-reference contents inside an XML document.

3.3. StorageResolver

A StorageResolver is used by KeyInfo and it's child objects / Elements to retrieve Certificates from storage locations. This approach is used to allow a user to customize the library for use in a specific corporate environment. It's possible to write StorageResolver s who make requests to LDAP servers or to use specificic PKI interfaces.
Bundled with the software come three sample StorageResolver s which can be used for common tasks:

- The KeyStoreResolver is able to retrieve Certificates from a JAVA KeyStore object. This KeyStoreResolver is constructed from an open JAVA KeyStore.
- The SingleCertificateResolver resolves only to a single Certificate. The SingleCertificateResolver is constructed using this single Certificate.
- The CertsInFilesystemDirectoryResolver is useful for resolving to raw X.509 certificates which reside as separate files in a directory in the filesystem. Such a resolver is needed for verifying the test signatures from Merlin Huges which are bundled in a directory.

StorageResolvers are supplied to the KeyInfo's addStorageResolver() method.

Generally, a StorageResolver has only a method to return an Iterator which iterates through the available Certificates.

3.3. KeyResolver

A KeyResolver is used by KeyInfo to process it's child Elements. There exist two general classes of a KeyResolver :

- If a ds:RSAKeyValue or ds:DSAKeyValue or ds:X509Certificate is used inside the ds:KeyInfo, the resolvers can return a public key or Certificate directly without further action, because the key itself is contained inside the ds:Signature.
- If there is only key material identification information like a ds:KeyName or the serial number of the Certificate, the KeyResolver must use the StorageResolvers to query the available keys and certificates to find the correct one.

Of course, there are cross-dependencies: e.g. a KeyResolver named RetrievalMethodResolver uses the ResourceResolver framework to retrieve a public key or certificate from an arbitrary location.

4. Secure Validation

A property was added in the 1.5.0 release to enable "secure validation". This property is true by default from the 2.3.0 release, but **false** for earlier releases. When set to true, it enforces the following processing rules:

- Limits the number of Transforms per Reference to a maximum of 5.
- Does not allow XSLT transforms.
- Does not allow a RetrievalMethod to reference another RetrievalMethod.
- Does not allow a Reference to call the ResolverLocalFilesystem or the ResolverDirectHTTP (references to local files and HTTP resources are forbidden).
- Limits the number of references per Manifest (SignedInfo) to a maximum of 30.
- MD5 is not allowed as a SignatureAlgorithm or DigestAlgorithm.
- Guarantees that the Dereferenced Element returned via Document.getElementById is unique by performing a tree-search.
- **1.5.6** Does not allow DTDs

This functionality is supported in the core library through additional method signatures which take a boolean, and in the JSR-105 API via the property "org.apache.jcp.xml.dsig.secureValidation, e.g.:

```
XMLValidateContext context = new DOMValidateContext(key, elem);
context.setProperty("org.apache.jcp.xml.dsig.secureValidation", Boolean.TRUE);
```

Secure Validation should be enabled in production as otherwise various attacks might be possible