XML Signature

Service Oriented Architectures

Module 1 - Basic technologies

Fulvio Frati

DTI - Università degli Studi di Milano

XML Signature

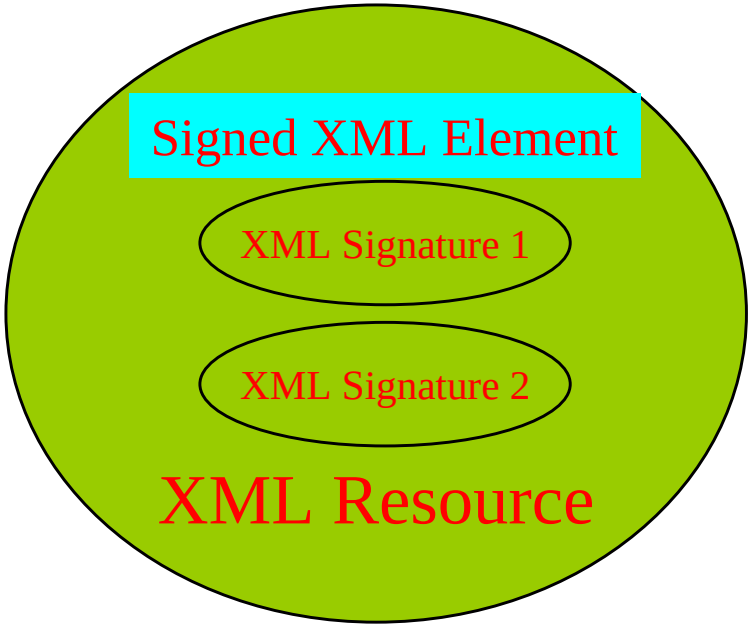
"XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type…"

W3C Recommendation 10 June 2008

Data objects are digested, the resulting value is placed in an element (with other information) and that element is then digested and cryptographically signed

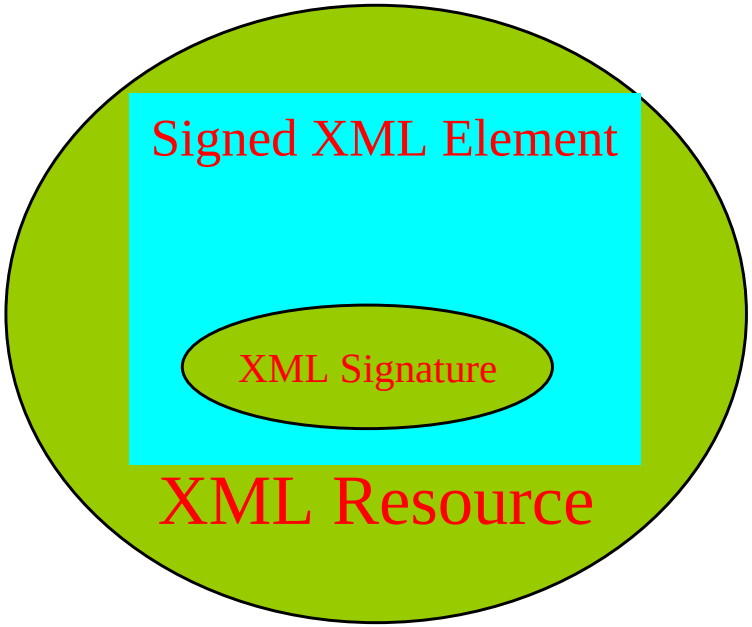
Signature Types - 1

Enveloped: the XML signature will itself be embedded within the signed document



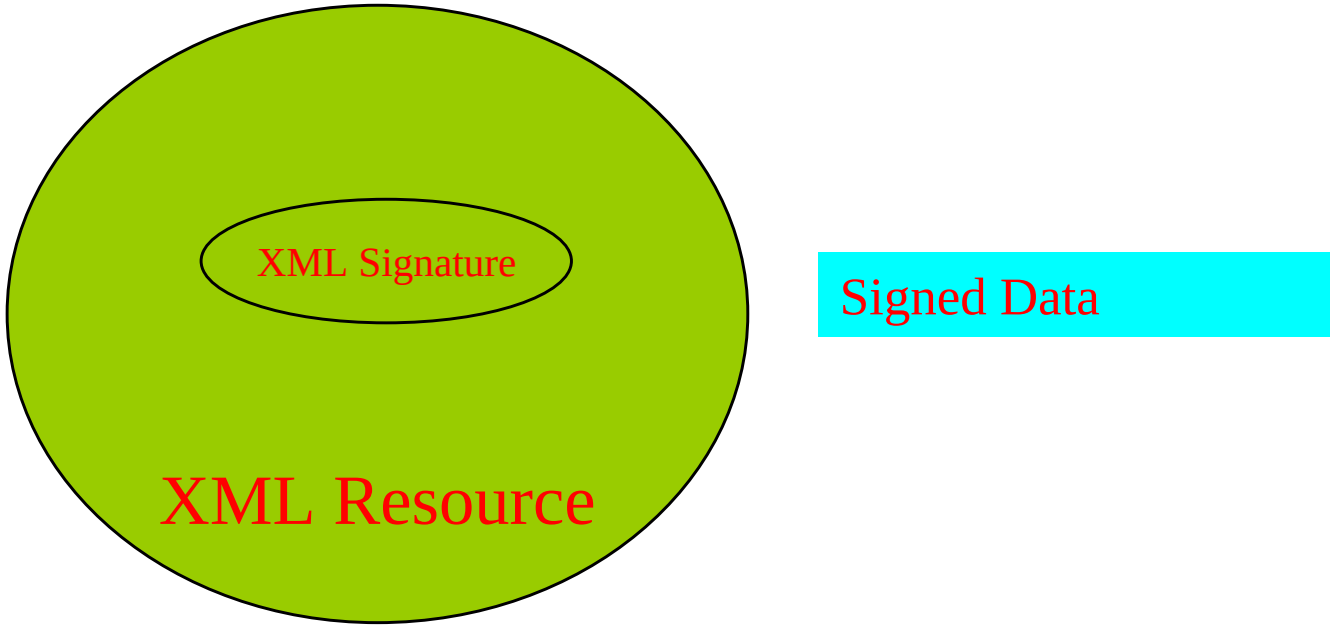
Signature Types - 2

Enveloping: the signed data is actually embedded within the XML signature element

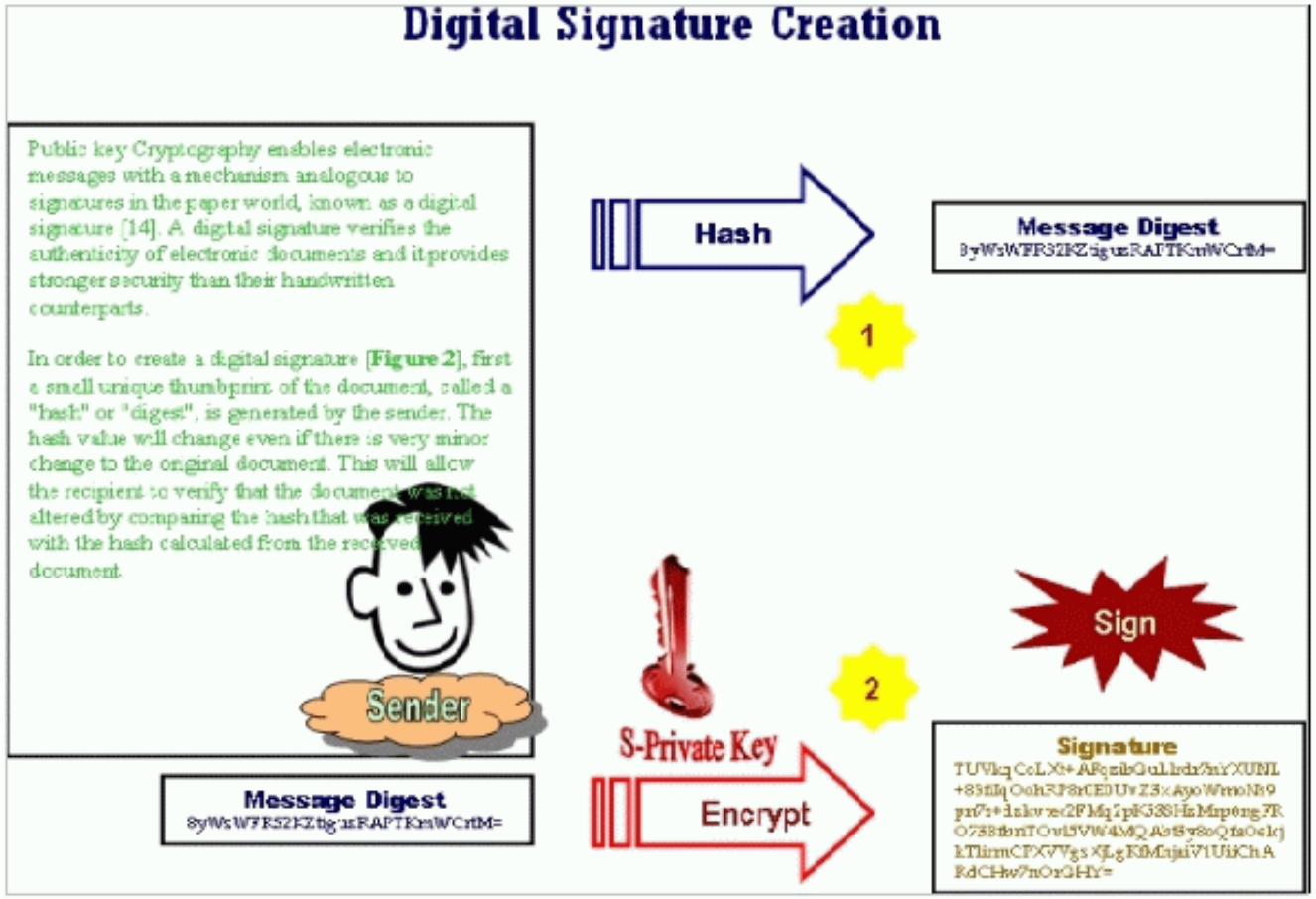


Signature Types - 3

Detached: the signed entities and the XML signature are separate



Signature Generation

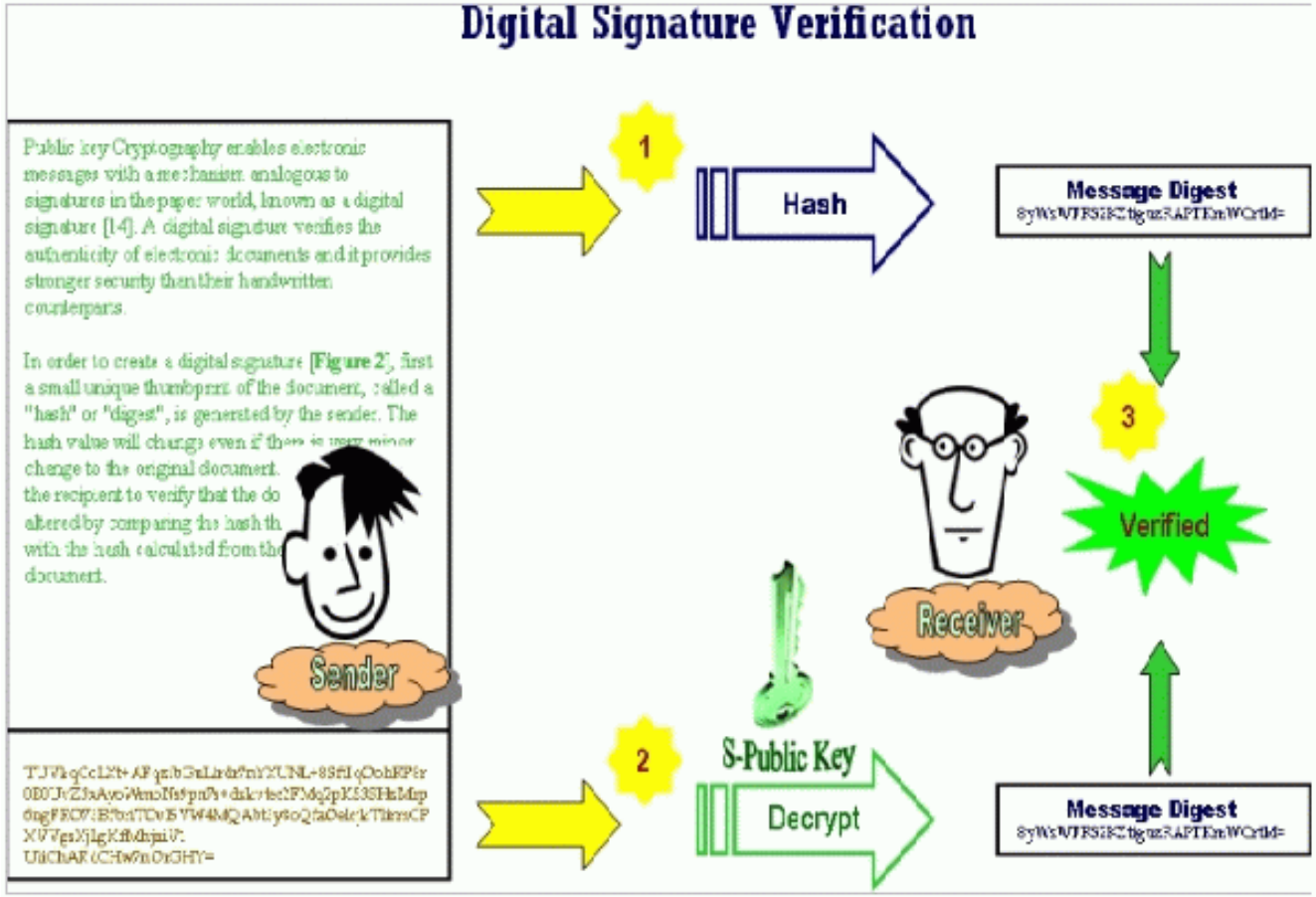


The sender generates a small unique a *hash* or *digest of the document*

Every minor change will cause the hash value to change

By comparing the hash that was received with the hash calculated from the received document, the recipient can verify whether the document was altered.

Signature Verification



The hash of the document signed or encrypted with the sender's private key acts as a digital signature for the document

The recipient will be able to verify or decrypt the signature by taking a hash of the message and verifying it with the signature that accompanied the message and the sender's public key.

Signature Scopes

The digital signature protocol helps to ensures the following:

- The signature is **authentic**.
  When the receiver verifies the message with the sender's public key, the receiver knows that the sender signed it.

- The signature **cannot be forged**.
  Only the sender knows his or her private key.

- The signature **is not reusable**.
  The signature is a function of the document and cannot be transferred to any other document.

- The signed document is **unalterable**.
  If there is any alteration to the document, the signature verification will fail at the receiver's end because the hash value will be recomputed and will differ from the original hash value.

- The signature **cannot be repudiated**.
  The sender cannot deny previous committed actions, and the receiver does not need the sender's help to verify the sender's signature.

Signed XML Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Envelope xmlns="urn:envelope">
  <data>
    Hello world!
  </data>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>DojWQ7QyswPfYLsWEaK8h7EqJRQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>chErWdiV85oi27bYtChygD6yLHticdAI2qXEnUMr2xU1MlStGtFN0A==</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P>
        <Q>li7dzDacuo67Jg7mtqEm2TRuOMU=</Q>
        <G>Z4Rxsnqc9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==</G>
        <Y>gOVbR8sY8t8/hvongKmzU4XRtRq5ZW6blQJT5pFkEOXLNn1waAD14HltCwukBbq+VlaLQaVc9n8qwa8B4F7TTQ==</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
  </Signature>
</Envelope>
```

Signature Tags: *Signature*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Envelope xmlns="urn:envelope">
  <data>
    Hello world!
  </data>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>DojWQ7QyswPfYLsWEaK8h7EqJRQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>chErWdiV85oi27bYtChygD6yLHticdAI2qXEnUMr2xU1MlStGtFN0A==</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P>
        <Q>li7dzDacuo67Jg7mtqEm2TRuOMU=</Q>
        <G>Z4RxsnqC9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==</G>
        <Y>gOVbR8sY8t8/hvongKmzU4XRtRq5ZW6blQJT5pFkEOXLNn1waAD14HltCwukBbq+VlaLQaVc9n8qwa8B4F7TTQ==</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
  </Signature>
</Envelope>
```

Root element of an XML Signature

Contains *SignedInfo*, *SignatureValue*, *KeyInfo (0/1)*, and *Object (0+)*

Signature Tags: *SignedInfo*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Envelope xmlns="urn:envelope">
  <data>
    Hello world!
  </data>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>DojWQ7QyswPfYLsWEaK8h7EqJRQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>chErWdiV85oi27bYtChygD6yLHticdAI2qXEnUMr2xU1MlStGtFN0A==</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P>
        <Q>li7dzDacuo67Jg7mtqEm2TRuOMU=</Q>
        <G>Z4RxsnqC9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==</G>
        <Y>gOVbR8sY8t8/hvongKmzU4XRtRq5ZW6blQJT5pFkEOXLNn1waAD14HltCwukBbq+VlaLQaVc9n8qwa8B4F7TTQ==</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
  </Signature>
</Envelope>
```

Includes

canonicalization algorithm, specifies the algorithm applied to prepare the document for signing

signature algorithm, specifies the algorithm used for signature generation and validation

transform algorithms, indicates how the signer obtained the data object that was digested

digest method, identifies the digest algorithm to be applied to the signed object

digest value, element that contains the encoded (base64) value of the digest

Signature Tags: *SignatureValue*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Envelope xmlns="urn:envelope">
  <data>
    Hello world!
  </data>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>DojWQ7QyswPfYLsWEaK8h7EqJRQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>chErWdiV85oi27bYtChygD6yLHticdAI2qXEnUMr2xU1MlStGtFN0A==</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P>
        <Q>li7dzDacuo67Jg7mtqEm2TRuOMU=</Q>
        <G>Z4RxsnqC9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==</G>
        <Y>gOVbR8sY8t8/hvongKmzU4XRtRq5ZW6blQJT5pFkEOXLNn1waAD14HltCwukBbq+VlaLQaVc9n8qwa8B4F7TTQ==</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
  </Signature>
</Envelope>
```

Contains the actual value of the digital signature

It is always encoded using base64

Signature Tags: *KeyInfo*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Envelope xmlns="urn:envelope">
  <data>
    Hello world!
  </data>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>DojWQ7QyswPfYLsWEaK8h7EqJRQ=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>chErWdiV85oi27bYtChygD6yLHticdAI2qXEnUMr2xU1MlStGtFN0A==</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P>
        <Q>li7dzDacuo67Jg7mtqEm2TRuOMU=</Q>
        <G>Z4Rxsnqc9E7pGknFFH2xqaryRPBaQ01khpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==</G>
        <Y>gOVbR8sY8t8/hvongKmzU4XRtRq5ZW6blQJT5pFkEOXLNn1waAD14HltCwukBbq+VlaLQaVc9n8qwa8B4F7TTQ==</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
  </Signature>
</Envelope>
```

Indicates the method to obtain the key needed to validate the signature

keyName, contains a string identifier related to the key pair used to sign the message

keyValue, contains a single public key that may be useful in validating the signature

DSAKeyValue, specifies fields for encription/decription based on DSA Signature Algorithm

X509Data, contains one or more identifiers of keys or X509 certificates (or certificates' identifiers or a revocation list)

RetrievalMethod, convey a reference to KeyInfo information that is stored at another location

C & Java Frameworks

Several implementation of W3C recommendations are available in most common programming languages

We focus on:

- **XMLSec C Library**
- **Java XML Digital Signature API (javax.crypto package)**

C - XMLSec

XML Security Library (XMLSec) provides support for XML Digital Signature and XML Encryption.

It is based on LibXML/LibXSLT and can use practicaly any crypto library (currently support for OpenSSL, GnuTLS and NSS).

XMLSec: Initialize Libraries

```
/* Init libxml and libxslt libraries */
xmlInitParser();
LIBXML_TEST_VERSION
xmlLoadExtDtdDefaultValue = XML_DETECT_IDS | XML_COMPLETE_ATTRS;
xmlSubstituteEntitiesDefault(1);

/* Init xmlsec library */
if(xmlSecInit() < 0) {
            fprintf(stderr, "Error: xmlsec initialization failed.\n");
            return(-1);
}

/* Init crypto library */
if(xmlSecCryptoAppInit(NULL) < 0) {
            fprintf(stderr, "Error: crypto initialization failed.\n");
            return(-1);
}

/* Init xmlsec-crypto library */
if(xmlSecCryptoInit() < 0) {
            fprintf(stderr, "Error: xmlsec-crypto initialization failed.\n");
            return(-1);
}
```

XMLSec: Sign File - 1

```
/* load template */
  doc = xmlParseFile(tmpl_file);
  if ((doc == NULL) || (xmlDocGetRootElement(doc) == NULL)){
              fprintf(stderr, "Error: unable to parse file \"%s\"\n", tmpl_file);
              goto done;
  }

  /* find root node in the document to sign and retrieve signature parameters directly from XML */
  node = xmlSecFindNode(xmlDocGetRootElement(doc), xmlSecNodeSignature, xmlSecDSigNs);
  if(node == NULL) {
              fprintf(stderr, "Error: start node not found in \"%s\"\n", tmpl_file);
              goto done;
  }

  /* create signature context */
  dsigCtx = xmlSecDSigCtxCreate(NULL);
  if(dsigCtx == NULL) {
    fprintf(stderr,"Error: failed to create signature context\n");
              goto done;
  }
```

XMLSec: Sign File - 2

```
/* load private key, assuming that there is not password – private key file (*.pem) supplied by user */
  dsigCtx->signKey = xmlSecCryptoAppKeyLoad(key_file, xmlSecKeyDataFormatPem, NULL, NULL, NULL);
  if(dsigCtx->signKey == NULL) {
     fprintf(stderr,"Error: failed to load private pem key from \"%s\"\n", key_file);
                goto done;
  }

  /* set key name to the file name, this is just an example! */
  if(xmlSecKeySetName(dsigCtx->signKey, key_file) < 0) {
                fprintf(stderr,"Error: failed to set key name for key from \"%s\"\n", key_file);
                goto done;
  }

  /* sign the template */
  if(xmlSecDSigCtxSign(dsigCtx, node) < 0) {
     fprintf(stderr,"Error: signature failed\n");
                goto done;
  }

  /* print signed document to stdout */
  xmlDocDump(stdout, doc);

  /* success */
  res = 0;
```

XML Document to Sign

```
<?xml version="1.0" encoding="UTF-8"?>
<!-
XML Security Library example: Simple signature template file for sign1 example.
-->
<Envelope xmlns="urn:envelope">
    <Data>
        Hello, World!
    </Data>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm=
                "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            <SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1" />
    <DigestValue></DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue/>
        <KeyInfo>
            <KeyName/>
        </KeyInfo>
    </Signature>
</Envelope>
```

Data to Sign

Signature Parameters

Signed XML Document

```
<Envelope xmlns="urn:envelope">

    <Data>
        Hello, World!
    </Data>

    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
            <CanonicalizationMethod Algorithm=
                "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            <SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
            <Reference URI="">
                <Transforms>
                    <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                </Transforms>
                <DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1" />

                <DigestValue>9H/rQr2Axe9hYTV2n/tCp+3UIQQ=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>Mx4psIy9/UY+u8QBJRDrwQWKRaCGz0WOVftyDzAe6WHAFSjMNr7qb2ojq9kdipT8
                        Oub5q2OQ7mzdSLiiejkrO1VeqM/90yEIGI4En6KEB6ArEzw+iq4N1wm6EptcyxXx
                        M9StAOOa9ilWYqR9Tfx3SW1urUIuKYgUitxsONiUHBVaW6HeX51bsXoTF++4ZI+D
                        jiPBjN4HHmr0cbJ6BXk91S27ffZIfp1Qj5nL9onFLUGbR6EFgu2luiRzQbPuM2tP
                        XxyI7GZ8AfHnRJK28ARvBC9oi+O1ej20S79CIV7gdBxbLbFprozBHAwOEC57YgJc
                        x+YEjSjcO7SBIR1FiUA7pw==
        </SignatureValue>
        <KeyInfo>
            <KeyName>rsakey.pem</KeyName>
        </KeyInfo>

    </Signature>
</Envelope>
```

Signature Values

XMLSec: Verify - 1

```
 /* load file */
 doc = xmlParseFile(xml_file);
 if ((doc == NULL) || (xmlDocGetRootElement(doc) == NULL)){
     fprintf(stderr, "Error: unable to parse file \"%s\"\n", xml_file);
 goto done;
}

/* find root node in the document to sign and retrieve signature parameters directly from XML */
node = xmlSecFindNode(xmlDocGetRootElement(doc), xmlSecNodeSignature, xmlSecDSigNs);
if(node == NULL) {
     fprintf(stderr, "Error: start node not found in \"%s\"\n", xml_file);
 goto done;
}

/* create signature context */
dsigCtx = xmlSecDSigCtxCreate(NULL);
if(dsigCtx == NULL) {
     fprintf(stderr,"Error: failed to create signature context\n");
 goto done;
}

/* load public key - private key file (*.pem) supplied by user */
dsigCtx->signKey = xmlSecCryptoAppKeyLoad(key_file, xmlSecKeyDataFormatPem, NULL, NULL, NULL);
if(dsigCtx->signKey == NULL) {
     fprintf(stderr,"Error: failed to load public pem key from \"%s\"\n", key_file);
 goto done;
}
```

XMLSec: Verify - 2

```
/* set key name to the file name, this is just an example! */
    if(xmlSecKeySetName(dsigCtx->signKey, key_file) < 0) {
        fprintf(stderr,"Error: failed to set key name for key from \"%s\"\n", key_file);
        goto done;
    }

  /* Verify signature */
    if(xmlSecDSigCtxVerify(dsigCtx, node) < 0) {
        fprintf(stderr,"Error: signature verify\n");
        goto done;
    }

  /* print verification result to stdout */
    if(dsigCtx->status == xmlSecDSigStatusSucceeded) {
        fprintf(stdout, "Signature is OK\n");
        } else {
          fprintf(stdout, "Signature is INVALID\n");
        }
```

XMLSEC Practical Example

Java Digital XML Signature APIs

Provides support for various implementations of digital signature algorithms and transforms as specified by W3C's XML-signature syntax and processing specification

The **javax.xml.crypto** package contains common classes that are used to perform XML cryptographic operations such as generating and verifying XML signatures

Java - Signature Generation - 1

Generate an enveloped XML Signature using the XML Digital Signature API

1- Use a JAXP *DocumentBuilderFactory* to parse the XML document to sign

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
```

2- Get an instance of a *DocumentBuilder*, used to parse the document

```
DocumentBuilder builder = dbf.newDocumentBuilder();
Document doc = builder.parse(new FileInputStream(argv[0]));
```

Java - Signature Generation - 2

3- Create a DSA KeyPair with a length of 512 byte (only for this example);

- the private key is usually previously generated and stored in a KeyStore file with an associated public key certificate

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
kpg.initialize(512);
KeyPair kp = kpg.generateKeyPair();
```

4- Create an XML Digital Signature *XMLSignContext* containing input parameters for generating the signature

- two parameters, the private key that will be used to sign the document and the root of the document to be signed

```
DOMSignContext dsc =
      new DOMSignContext (kp.getPrivate(), doc.getDocumentElement());
```

Java - Signature Generation - 3

5- assemble the different parts of the *Signature* element into an *XMLSignature* object

```
XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM")
```

6- Add the *Reference* part passing the document to sign, the digest method, the type of transform

```
Reference ref = fac.newReference
  ("", fac.newDigestMethod(DigestMethod.SHA1,null),
  Collections.singletonList fac.newTransform(Transform.ENVELOPED,
 (TransformParameterSpec) null)), null, null);
```

7- create the *SignedInfo* object passing the *Canonicalization* method, the *Signature* method, and the References

```
SignedInfo si = fac.newSignedInfo(fac.newCanonicalizationMethod
  (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,(C14NMethodParameterSpec) null),
  fac.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
  Collections.singletonList(ref));
```

Java - Signature Generation - 4

8- Create the optional *KeyInfo* object

- Enables the recipient to find the key needed to validate the signature

```
KeyInfoFactory kif = fac.getKeyInfoFactory();
KeyValue kv = kif.newKeyValue(kp.getPublic());
KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));
```

9- Create the *XMLSignature* object, passing as parameters the *SignedInfo* and *KeyInfo* objects

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

Java - Signature Validation - 1

1- Use a JAXP *DocumentBuilderFactory* to parse the XML document containing the Signature

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
```

2- Get an instance of a *DocumentBuilder* to parse the document

```
DocumentBuilder builder = dbf.newDocumentBuilder();
Document doc = builder.parse(new FileInputStream(argv[0]));
```

3- Specify the *Signature* element to validate passing the XML Signature namespace URI and the tag name of the *Signature* element

```
NodeList nl = doc.getElementsByTagNameNS(XMLSignature.XMLNS, "Signature");
if (nl.getLength() == 0) {
        throw new Exception("Cannot find Signature element");
}
```

Java - Signature Verification - 2

4- Create an *XMLValidateContext* instance for validating the signature

- Parameters: a *KeyValueKeySelector* object and a reference to the Signature element to be validated

```
DOMValidateContext valContext =
        new DOMValidateContext (new KeyValueKeySelector(), nl.item(0));
```

5- Extract the contents of the Signature element (*unmarshalling*) into an *XMLSignature* object

```
XMLSignatureFactory factory = XMLSignatureFactory.getInstance("DOM");
XMLSignature signature = factory.unmarshalXMLSignature(valContext);
```

6- Validate the signature

```
boolean coreValidity = signature.validate(valContext);
```

Java - Signature Verification - 3

If the *XMLSignature.validate* method returns `false`, we can try to narrow down the cause of the failure

- **Signature validation**: the cryptographic verification of the signature

```
boolean sv = signature.getSignatureValue().validate(valContext);
```

- **Reference validation**: the verification of the digest of each reference in the signature

```
boolean refValid = ((Reference) i.next()).validate(valContext);
```

Java Practical Example

References

W3C XML Signature Syntax and Processing

http://www.w3.org/TR/xmldsig-core/

XML Digital Signature API

http://java.sun.com/javase/6/docs/technotes/guides/security/xmldsig/XMLDigitalSignature.html

XML Security Library

http://www.aleksey.com/xmlsec/