Providing Integrity and Confidentiality with the XML Security: Digital Signature and XML Encryption overview and usage examples

Draft version 0.1. - January 13, 2005

Yuri Demchenko <demch@science.uva.nl>

Abstracts

This document provides general overview of the XML Digital Signature and XML Encryption functionality and their use for providing document and message Integrity and Confidentiality for XML and Web Services based applications. Provided examples demonstrate using XML Signature and XML Encryption for and together with XACML Request/Response messages and SAML Assertions. Test programs are parts of the AAAuthreach package that provides simple API to Apache XML Security suite.

1 Introduction

XML Digital Signature and XML Encryption are two standard mechanisms used to provided integrity and confidentiality of XML documents and messages, in particular. Both of these mechanisms are integrated into SAML 2.0 and also used in SAML 1.1. Generally, XML Signature and XML Encryption can be added to any other XML format at the level of XML schema extension or to any valid XML document instance by using numerous available XML Security packages.

2 XML Digital Signature format and processing

This section provides general information about XML Signature format and some suggestions how it can be used together with SAML, XACML and XML Web Services in typical XML Web Services (WSA) and Open Collaborative Environment (OCE) applications.

2.1 XML Digital Signature format and processing

XML Signature has the following structure (see also picture below):

```
<Signature ID?>
       <SignedInfo>
       <CanonicalizationMethod/>
       <SignatureMethod/>
       (<Reference URI? >
              (<Transforms>)?
              <DigestMethod>
              <DigestValue>
       </Reference>)+
</SignedInfo>
       <SignatureValue>
(<KeyInfo>)?
(<Object ID?>)*
</Signature>
```

There are three types of the XML Signature algorithms:

- Enveloped XML Signature.
  An enveloped signature is a signature of either an entire document or a document fragment, where the XML signature will itself be embedded within the signed document. An enveloped signature transform is always used to remove the signature structure from the signing process.

- Enveloping XML Signature
  An enveloping signature is a signature where the signed data is actually embedded within the XML signature structure: The XML signature specification provides the ability for arbitrary XML structures to be embedded within a signature, for this express purpose.

- Detached XML Signature
  A detached signature is a signature where the signed entities are separate from the actual signature fragment. The signed entities can be remote XML documents or remote non-XML documents. They can also be XML fragments located elsewhere in the same document as the XML signature.

It is important to mention that WS-I Basic Security Profile for Web Services recommends using detached signature and strongly discourages enveloping signature use, enveloped signature can be used but it provides limited functionality for signing the document and managing security components.

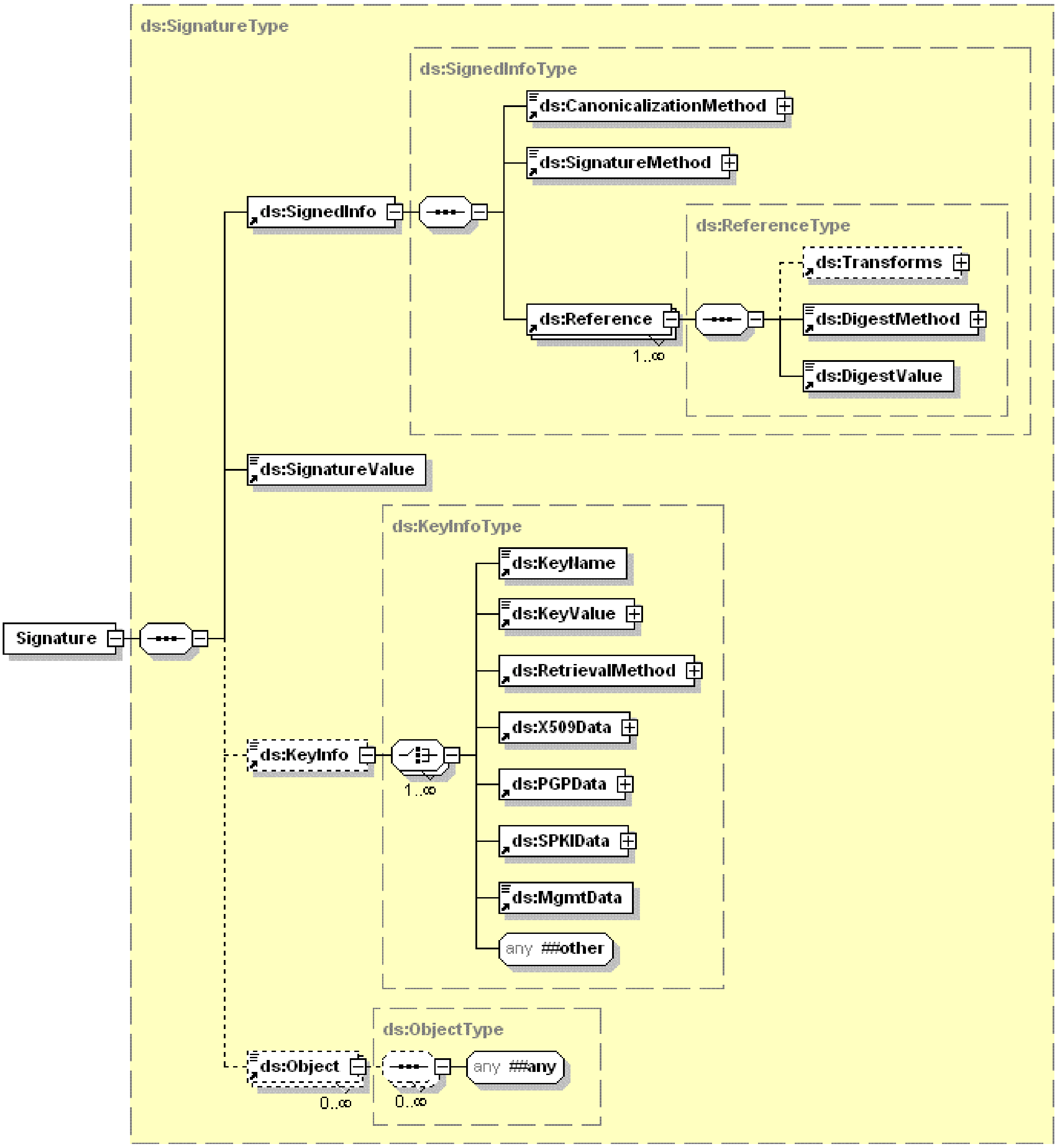

Figure 1. XML Digital Signature structure

Creation of XML Signature includes the following steps:

1. Determine which resources are to be signed
2. Calculate the digest of each resource
3. Collect the Reference elements
4. Signing
5. Add key information
6. Enclose in a Signature element

XML Signatures verification includes the following steps:

- Verify the signature of the `<SignedInfo>` element
- Recalculate the digest of the `<SignedInfo>` element (using the digest algorithm specified in the `<SignatureMethod>` element)

- Use the public verification key to verify that the value of the <SignatureValue> element is correct for the digest of the <SignedInfo> element.
  If this step passes
- Recalculate the digests of the references contained within the <SignedInfo> element and compare them to the digest values expressed in each <Reference> element's corresponding <DigestValue> element. XML Signature standard defines a number of transformations

required for XML Signature signing and validation:

- Canonicalisation
- Base64
- XPath Filtering
- Envelope Signature Transform

XSLT TransformationA canonicalisation is required to ensure the normalised presentation of the XML document in cases when XML format and schema validation allows options, like in using character encoding or attributes ordering. The ***canonical form*** of an XML document is physical representation of the document produced by the canonicalisation method that implies the following changes.

1) Encoding and characters

- The document is encoded in UTF-8Line breaks normalized to #xA on input, before parsing
- Whitespace outside of the document element and within start and end tags is normalized
- All whitespace in character content is retained (excluding characters removed during line feed normalization)

2) Elements and references

- Character and parsed entity references are replaced
- CDATA sections are replaced with their character content
- The *XML declaration* and *document type declaration* (DTD) are removed
- Empty elements are converted to start-end tag pairs

3) Attributes

- Attribute values are normalized, as if by a *validating processor*Attribute value delimiters are set to quotation marks (double quotes)
- Special characters in attribute values and character content are replaced by character references
- Superfluous namespace declarations are removed from each element
- Default attributes are added to each element

Lexicographic order is imposed on the namespace declarations and attributes of each elementXML canonicalisation is defined in terms of the XPath definition of a node-set. If an XML document must be converted to a node-set, XPath REQUIRES that an XML processor be used to create the nodes of its data model to fully represent the document. The XML processor performs the following tasks in order:

- normalize line feeds
- normalize attribute values
- replace CDATA sections with their character content
- resolve character and parsed entity references

The input octet stream MUST contain a well-formed XML document, but the input need not be validated. The declarations in the document type declaration are used to help create the canonical form. This is a positive feature of the XML Signature transformation that signing program doesn't need to validate a document.

2.2 XML Digital Signature examples

Examples below contain an Enveloped XML Digital Signature that signed the whole XML document URI = “” and the Subject element URI = “#subject” preserving in this way the document integrity as whole and the element that contains security token.

```
<AAARequest xmlns="urn:oasis:names:tc:xacml:1.0:context"
xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="xmlns
data/schemas/aaa-cnl-msg-xacml-01-nons-dsig.xsd">
  <Subject Id="subject" SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="admin@gaaa.collaboratory.nl">
      <AttributeValue>WHO740@users.collaboratory.nl</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:token"
DataType="http://www.w3.org/2001/XMLSchema#string" Id="subjecttoken"
Issuer="admin@gaaa.collaboratory.nl">
      <AttributeValue>2SeDFGVHYTY83ZXxEdsweOP8Iok)yGHxVfHom90</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:job-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="admin@gaaa.collaboratory.nl">
      <AttributeValue>JobID-XPS1-212</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="admin@gaaa.collaboratory.nl">
      <AttributeValue>customer</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="admin@gaaa.collaboratory.nl">
<AttributeValue>http://xps1.resources.collaboratory.nl/Phillips_XPS1</AttributeValu
e>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="admin@gaaa.collaboratory.nl">
      <AttributeValue>ControlExperiment</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="admin@gaaa.collaboratory.nl">
      <AttributeValue>ViewExperiment</AttributeValue>
    </Attribute>
  </Action>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
```

```xml
          <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315#WithComments"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>Hys8QOitLjUTasVkV7pS0S0UL6Q=</ds:DigestValue>
      </ds:Reference>
      <ds:Reference URI="#subject">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
          <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315#WithComments"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>2DLg3ZBWA2j971Mq0hyZAv29Y3w=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
deN8Ot8pfeUTC3zipgnGyW2v/oSYHKfCA2mU0Q8HRevbt0eaAxoQh/VP1r2jst8M6Uw9NyVWdQG6
Pp2XVH5VOdv2f3+m7X+1z18GjHrBQOhfsqpT96W+Vjhv2cAgS09GEBNFSqO7gZMJ/ix+kjiqWp1O
DUNRuKF6fNmxHxanMNo=
</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICADCCAWkCBEGX/FYwDQYJKoZIhvcNAQEEBQAwRzELMAkGA1UEBhMCTkwxGTAXBgNVBAoTEENv
bGxhYm9yYXRvcnkubmwxHTAbBgNVBAMTFEFBQXV0aHJlYWNoIFNlY3VyaXR5MB4XDTA0MTExNTAw
NDYxNFoXDTA1MDIxMzAwNDYxNFowRzELMAkGA1UEBhMCTkwxGTAXBgNVBAoTEENvbGxhYm9yYXRv
cnkubmwxHTAbBgNVBAMTFEFBQXV0aHJlYWNoIFNlY3VyaXR5MIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQDdDrBhVmr1nD9eqi7U7m4yjIRxfvjAKv33EpuajvTKHpKUgLjbcBC3jNJ4F7a0GiXQ
cVbuF/aDx/ydIUJXQktvFxK0Sm77WVeSel0cLc1hYfUSAg4mudtfsB7rAj+CzNnVdr6RLFpS9YFE
lv5ptGaNGSbwHjU02HnArEGL2K+0AwIDAQABMA0GCSqGSIb3DQEBBAUAA4GBADHKqkOW4mP9DvOi
bMvf4oqXTth7yv8o3Zol7+nqlB9Tqf/bVNLMk8vNo5fWRHbpnHIFFgTk31nrJf8kEZEofvwAeW9s
1gQtYfs1oxvsMPKHxFjJDiZlLkHRViJl/slz5a7pkLqIXLRsPFRziTksemRXB/fT8KDzM14pzQZg
HicO
</ds:X509Certificate>
      </ds:X509Data>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
3Q6wYVZq9Zw/Xqou1O5uMoyEcX74wCr99xKbmo70yh6SlIC423AQt4zSeBe2tBol0HFW7hf2g8f8
nSFCV0JLbxcStEpu+1lXknpdHC3NYWH1EgIOJrnbX7Ae6wI/gszZ1Xa+kSxaUvWBRJb+abRmjRkm
8B41NNh5wKxBi9ivtAM=
</ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
  </ds:Signature>
</AAARequest>
```

Listing 1. Example Enveloped XML Signature signed with the RSA private key and containing RSA public key.

Example below shows the <ds:KeyInfo> format for the DSA public key:

```xml
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICnTCCAlsCBEG8RI8wCwYHKoZIzjgEAwUAMDQxCzAJBgNVBAYTAk5MMQ8wDQYDVQQKEwZNYXRy
aXgxFDASBgNVBAMTC0FnZW50IFNtaXRoMB4XDTA0MTIxMjEzMTU1OVoXDTA1MDMxMjEzMTU1OVow
NDELMAkGA1UEBhMCTkwxDzANBgNVBAoTBk1hdHJpeDEUMBIGA1UEAxMLQWdlbnQgU21pdGgwggG4
MIIBLAYHKoZIzjgEATCCAR8CgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAi
UftZPY1Y+r/F9bow9subVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYV
DwT7g/bTxR7DAjVUE1oWkTL2dfOuK2HXKu/yIgMZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc
9QKBgQD34aCF1ps93su8q1w2uFe5eZSvu/o66oL5V0wLPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPo
TCgWE7fPCTKMyKbhPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa
5Z8GkotmXoB7VSVkAUw7/s9JKgOBhQACgYEAk9nKczg5fwvvj/97LLW5lVXw45ojopRw1vsRiejo
hKoUMpd8QO6okIkqUDWijMWa208Z6Z++UE7dy1XOpG8vjso1xq4dGfNyCDYth045hkK+qoN1DRA2
S2Hagt3ZfR4PL4n9TDdJkZaCM0H+gZx8URG2v+7vymuvbMdvYvYYcwgwCwYHKoZIzjgEAwUAAy8A
MCwCFF54S0t1s9vEQBniu5R25n4lxNpwAhQia4j/upIpdLVBdfpxLk5X5qoLQQ==
```

```
</ds:X509Certificate>
      </ds:X509Data>
      <ds:KeyValue>
        <ds:DSAKeyValue>
          <ds:P>
/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuA
HTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVUE1oWkTL2dfOu
K2HXKu/yIgMZndFIAcc=
</ds:P>
          <ds:Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</ds:Q>
          <ds:G>
9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3
zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSo=
</ds:G>
          <ds:Y>
k9nKczg5fwvvj/97LLW5lVXw45ojopRw1vsRiejohKoUMpd8QO6okIkqUDWijMWa208Z6Z++UE7d
y1XOpG8vjso1xq4dGfNyCDYth045hkK+qoN1DRA2S2Hagt3ZfR4PL4n9TDdJkZaCM0H+gZx8URG2
v+7vymuvbMdvYvYYcwg=
</ds:Y>
        </ds:DSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
```


Listing 2. Fragment <ds:KeyInfo> element with the DSA public key.

2.3 XML Signature Implementation suggestions for WSA/OCE:

For the purpose of WSA/OCE security assertions/tickets exchange and managing security context in the document-centric security model, the enveloped XML Signature provides necessary functionality:

- Enveloped signature is always placed under the root of document, however multiple Signatures are allowed in one document.

- Enveloped Signature can sign document in total and multiple elements defined by multiple Reference elements

- To protect XML document integrity, it is recommended to sign the whole document with URI = “” and other critical security components such as security tokens, subject attributes and conditions that can identified by element’s ID/Id, xpointer or XPath expressions.

- Such voluminous component of the XML Signature element as KeyInfo can be reduced to KeyName if exchanging systems use pre-installed shared keys or public key certificates.

- The input document (in the form of octet stream or XML object) MUST be well-formed XML document, but the input needs not to be validated.

- XMLSig can be easily combined with the XML document validation what is highly rfecommended from the security point of view, if we are using the Enveloped XMLSig or providing a single container for the <ds:Signature> elements in the Detached XMLSig. The XMLSig elements or elements can ignored during the validation process, or added to the XML document schema.

- When considering use of one of available open source packages, it is important to take into account the following:

  (1) IBM’s xss4j was popular at the early stage of XML Security dissemination but now it is announced not to be developed further;

(2) Apache XML Security suite is widely used, including Sun's XML Web Services Development Pack and Globus Toolkits 3.x

(3) Java XML Digital Signature API (JSR 105) potentially may become the most preferable package for processing XML Digital Signature as it implements Java community standard, it provides rich functionality and strict standards implementation, but its use currently limited by limited number of usage examples; another limitation is that it is not combined with the XML Encryption API that is being developed in the frame of JSR 106: XML Digital Encryption APIs but is not supported by any implementation.

3 XML Encryption

3.1 XML Encryption format and processing

XML Encryption datamodel:

```
<EncryptedData Id? Type? MimeType? Encoding?>
   <EncryptionMethod/>?
   <ds:KeyInfo>
      <EncryptedKey>?            # extension to XMLSig KeyInfo
      <AgreementMethod>?
      <ds:KeyName>?
      <ds:RetrievalMethod>?
      <ds:*>?       #
   </ds:KeyInfo>?
   <CipherData>                  # envelopes or references the raw encrypted data
      <CipherValue>?
      <CipherReference URI?>?    # points to the location of the raw encrypted data
   </CipherData>
   <EncryptionProperties>?       # e.g., timestamp
</EncryptedData>
```

`CipherData` element contains the encrypted octet sequence as base64 encoded text of the CipherValue element, or provides a reference to an external location containing the encrypted octet sequence via the CipherReference element.

```
<element name='CipherData' type='xenc:CipherDataType'/>
   <complexType name='CipherDataType'>
      <choice>
         <element name='CipherValue' type='base64Binary'/>
         <element ref='xenc:CipherReference'/>
      </choice>
</complexType>
```

Figure 2 illustrates EncryptedData element structure. XML Encryption reuses ds:KeyInfo and ds:Transform elements from the XML Signature definition.

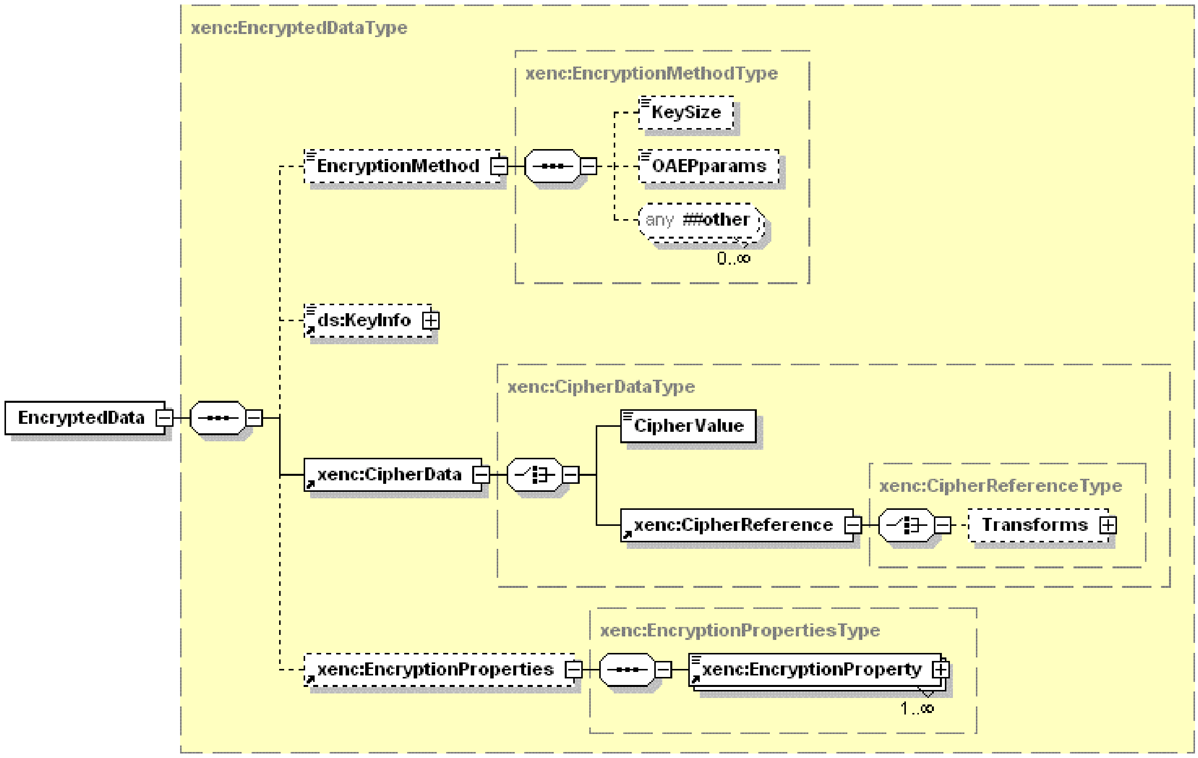

Figure 2. XML Encryption EncryptedData element structure

When encrypting data, for each `EncryptedData` and `EncryptedKey` the encryptor must:

1. Select the algorithm (and parameters)
2. Obtain and (optionally) represent the key
3. Encrypt the data
    - If the data is an "element" or element "content", obtain the octets by serialising the data in UTF-8; any other data must be serialised as octets
    - Encrypt the octets using the algorithm and key from steps 1 and 2
    - Provide type of presentation to indicate how to obtain and interpret the plaintext octets after decryption (e.g., MimeType="text/xml" or MimeType="image/png")
4. Build the EncryptedType (`EncryptedData` or `EncryptedKey`)
5. Process EncryptedData
    - If the Type of the encrypted data is "element" or element "content", then encryptor SHOULD be able to replace the unencrypted "element" or "content" with the EncryptedData element.
    - If the Type of the encrypted data is "element" or element "content", then encryptor MUST always be able to return the EncryptedData to the application.

Decryption processing rules include the following steps:

1. Locate encrypted data container xenc:EncryptedData that must have an attribute "type" that indicate may replace the element or
2. Process the element to determine the algorithm, parameters and ds:KeyInfo element to be used. If some information is omitted, the application MUST supply it.
3. Locate the data encryption key according to the ds:KeyInfo element, which may contain one or more children elements.
4. Decrypt the data contained in the CipherData element – depending on existence of CipherValue or CipherReference child elements

5. Process decrypted data of Type 'element' or element 'content'
   - The `cleartext` octet sequence (from step 3) is interpreted as UTF-8 encoded character data
   - The decryptor MUST be able to return the value of Type and the UTF-8 encoded XML character data. Validation on the serialized XML is NOT REQUIRED.
   - The decryptor SHOULD support the ability to replace the EncryptedData element with the decrypted 'element' or element 'content' represented by the UTF-8 encoded characters
6. Process decrypted data if Type is unspecified or is *not* 'element' or element 'content'.

XML Encryption implementation in Apache XML Security package require some additional JCE providers that are not normally shipped with the standard JSDK1.4. In this case, we need to use third party JCE (which is recommended from BouncyCastle at http://www.bouncycastle.org/download/jce-jdk13-114.jar). More about Java Cryptographic Architecture see Appendix E.

The XML Encryption specification lists a number of required and optional cryptographic algorithms including:

- Block encryption – TripleDES, AES;
- Stream encryption;
- Key transport using public key encryption algorithms – RSA version 1.5, RSA-OAEP (Optimal Asymmetric Encryption Padding with RSA);
- Key agreement used for derivation of a shared secret key based on a shared secretcomputed from certain types of compatible public keys from both the sender and the recipient – Diffie-Helman key value, Diffie-Helman key agreement,
- Symmetric key wrap – Triple DES, AES;
- Message digest used in key agreement method as a part of key derivation, with PSA-OAEP and with HMAC message authentication code – SHA1, SHA256, SHA512, RIPEMD-160;
- Message authentication uses XML Signature method.

The Base64 encoding algorithm is used for binary encrypted data presentation. Every cipher must be Base64 encoded before it can be inserted into an XML document to be safely transferred over text based protocols.

3.2 XML Encryption examples

Examples below demonstrate encryption of the whole document (listing) content or one of element determined by its tag name, in our example the Subject element of the generated test AAA:Request message. an Enveloped XML Digital Signature that signed the whole XML document URI = “” and the Subject element URI = “#subject” preserving in this way the document integrity as whole and the element that contains security token.

```
<!-- Comment before -->
<AAA:AAARequest Id="CNLhashID#" cnl:attr1="CNL2test#" version="2.0" xmlns=""
xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">
  <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-
tripledes" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
        <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
```

```
          <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">98lmGOGqpYZR5KaeAmcphRPCIjiGAMGHwHu9
nokQ4/s=</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">z5Wa7EdgPVXSe6eLISUSiTICNPyp8fuBf3V+
IBXPTUMljy2BeVEz8qp9vQSuGKRYF5JzNyCnNra9
Mx140r9ZHrxUiWPJm2fWGZgnbOuHPpKN+J0Hatxm7xS021qhwLUIuAgrOX7pyR7QPhyfASG5JwLs
hLxot6YuarQODk/b2diN+a+HmgJC+FRuMoY68uZpNLL+1W2wOiXy/vLTA0w6lciMItxARWQfHXwD
v4h7371B2PciFDzQzqAE+C/DAtIi0Nyq4sk7Ng0wfY447v/OFmnrVwUZUWAzcOSfkMml74plBu2B
x6Hx0IFX6jZcebXMfZvqH0KbC1OQjQP2CT6VE8TgJ3qoMak11Wtzx1BotnyPlnl2c5sR30XOK565
Gb+gB31AjOFWPCUxMbcDas9f3Oeqcq9ickr5b3zwyPHrhzGonUitBaGHlwChLNJcLijmpbVeDVBZ
2ooXXc26TMxDWW4NHHL1N+/A64hecS0W5aEBhUozp5O4Es0CaT4IlxYPtarN2pnAZgSTZx2Fxc+/
/H62WakIc0vURT2gr8LbHifMKX9DyYp77auR1ZoVCTwNaQRGh9cjsR3ZVWOe6+hDefiYqntTzqFl
WDsybVbiKaWYH/j5z3HABdbrnALicrUuD19nqDyrN3t94/WQnQBRARZBRg05JrgMMCjsRaaYHMdS
6JuvruzMNmaKqpNuw/Hp60IavJ/R0IMFepLNkygFwSna91sNpHCUqQpKTPoM1F0=</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</AAA:AAARequest>
<!-- Comment after -->
```

Listing 3. Example XML Encryption of the content of the simplified AAA:Request message (using shared 3DES secret key for encryption of the data encryption key).

```
<!-- Comment before -->
<AAA:AAARequest Id="CNLhashID#" cnl:attr1="CNL2test#" version="2.0" xmlns=""
xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">
  <AAA:Subject Id="subject" xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">Subject element will contain the SubjectID,
AuthN token and Subject attributes
</AAA:Subject>
  <AAA:Resource xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">Resource element defines the target
resource
</AAA:Resource>
  <AAA:Action xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">Action element will contain the requested
action
</AAA:Action>
</AAA:AAARequest>
<!-- Comment after -->
```

Listing 4. Decrypted message that is equal to the original one.

```
<!-- Comment before -->
<AAA:AAARequest Id="CNLhashID#" cnl:attr1="CNL2test#" version="2.0" xmlns=""
xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">
  <AAA:Subject Id="subject" xmlns="" xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">
    <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-
tripledes" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
          <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">qij9zJgKZp9RiJxYN1QJAN0vhjLJSMGVLD/d
oQtmCsk=</xenc:CipherValue>
          </xenc:CipherData>
        </xenc:EncryptedKey>
      </ds:KeyInfo>
      <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
```

```
        <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">SVNtSy7q1PgxdJHlb4zphcL4hxSJIs3Aln/H
mZ1x8XvWL4rCZPnlslWLbg+fLwqQ66I3kN2StHPF
CuMMQ1u0i40JkKKTZYodFpuqQ/UhXZ2hzOazO5j8OMfATE6n9Urx</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </AAA:Subject>
  <AAA:Resource xmlns="" xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">Resource element defines the target
resource
</AAA:Resource>
  <AAA:Action xmlns="" xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">Action element will contain the requested
action
</AAA:Action>
</AAA:AAARequest>
<!-- Comment after -->
```

Listing 5. Example simplified AAA:Request message with the encrypted Subject element (the same encryption scenario).

```
<AAARequest xmlns="urn:oasis:names:tc:xacml:1.0:context"
xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="xmlns
data/schemas/aaa-cnl-msg-xacml-01-nons-dsig.xsd">
  <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-
tripledes" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
        <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">NxaCzuEaLizCB8nloGW3OVCKSt34Wfa/ObP4
MoJEuLA=</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">FDH/hy8HAYnGipI5ZkllzvQYCU3ghQ8+T1Jz
XsowqIuNvMSCNSfXulARJG2rOPuRu8eBn3vpFSUi
mI4R5/nJRkV3DSuhvurcTowr14vcRtYkXQwL7XKbpZLatNS5rECML1lkqqM2M/XCl9s75g3vIGF3
Ix4rGnSKHWP4CKFg3yTYqhQ7itIaTbvawOLwT/gKqVMB94tmJ2dgR4shi83z3xGTVWIygw1xnDJm
YdJ6mulE69oKgYPwm+g3ExsA3reRDhbhKRLmW2lmKybaNpWM4ePvlcv2mzYbvP8nqhkgRRL21Gvs
7V26HzMHCnvnJjLYLN4zeGF+l+a7S3y6XXFNvSJhIgeImC3DCbdRpMeYgZr2BSAhQvkSB85+gsce
/pXX5ZmFdUSXPGUiXreZ1pXYJiGlCITzITOzTf+z5/eCxcCv23An6MEaGkG7FZJNBoRhSKPvZbNe
W5YnaHeikx8BTqrUpcoVE8WpuiaMhxi6cdtwtXvsFy3gnKq/gUZEgw261ilOuBmobLQrM0t7G7Ut
HufuxubV5uSxKv389RUR0dulD9p6EWi3MUYlmdIgjVSWHZ/08MyYrnEvUOQJZ7ZAtvYMXc/BRe79
9WYoWGQ5qAhg2QS1odL/a080eK/YEEwIPm3FoR8HMf7ettx22gLeGIeNWmRIKjtinMAIA5d/C+GN
ICZVmggjlhhFIJ6LrVPBEv+XP+Q95/xFQASLcPZdl/SXgk6OHru4zuUCVEXYz68F4IkEVE201M4V
WG3yIOY4ZTwsbVTxad458N7hi9uKhtsnNSeN6OVsro94dt5Qn2RFvAgUXUXAueSbFYI9rTCquYC0
qGtL8ogmwJQBPd5yBvbp9z4+XjZqRxeuheK72K8FIYsma9cAtPsqszXtG4tACaNr2jO4r5NUwLpA
fked01gjcEyscTTH4uWaPfunSLqJGRn1IGJL99LM/T2MhNIUhmRlAeEwmdyK+PhZhsg71m4UNmB8
SO+RyONR+Xex+XyrOGNUuJVCsPIWR98IKx78sCPnkgvO7Lwu/LPPjV+7xvej0n6ErHCtZy7UIM/4
khvbHzX5o/5/whTWnuA9pTcwBKA7HfdzkwbCJBjw56BiLzR29RJMrlF6NKC4Gh/pdSkFIDas/J1m
oLTEKS/rnWwLqjzlcPA1O5Vix2IA9sRghT+6cuLf25gMFLZO2iZrSMlcHW6e86+K/I5oAmwv8HHG
E8viTtOkXPr6dYlusGZLJ635ElxXQYrW3g8mzolcPNExJjA2zG4HvaisetHXY32rAX9FV5Ht4K4r
mzE/INGFb4o8H+F/sk0YJEQwgHEQkVbPOWyqbfJz6Czdk35j5MdsonChVW2ZTE7Ec3IWqgNUjjrg
PtOpmgrPTNOmWVa9jU+yMvZAih6YAvKgzhOrtfXk7B567Tstg001n+v5eg1nJLCex0HhYQnRiari
PNi4myeZk+mUV+UFYyFIYcuOAx/DufJ15dfvyWRGpyGUjRQJXG1Ul2kiHz0sT3uUJm/THS2DdNa3
Ia/VMjzesuLQORgjHPi00d8R6/crUSbcJyzWbPy+19tQQCVfiIcHOukE+WdujOmAD63yArux+wqd
7jj1rlP8hreL7xxPdoDd8ugq3f4tMzpwf1i3rGbR8ESObv+wGshrwvbUhU/XbE2R5Dhrzuh1dcGQ
2BsRhheAT+8ReBLLbKsZxAtp/Em3eD3ja4Q92Y1H6rKWu48ZE9m6aL5LbrepeHP6ih9iomZ0ryAU
5RkyF9ObRJznt0+dmhdsZSHLG71RCvZqL4Ty284fZXIGrKwDddPYsrazU/BgMxx6vEs63+9divau
OxWuY+UiZEUsVfOEUzz7bNsu++y+eP5BDPhC/saIj2cy6PQoApcaC9M7uUorSGWtQBknLR/ltLzf
NBCbQbNdfnicf94s6tAtqOEP+arRY6P6e/QF/koXepYLzYhUBxAU+lozT3VXnOXUNc5HzQGiV2v4
4Lbwjcvzg/YD+9IdxDMNsIE4kSh68F27AzPdxU2J4P3PQO03CD0awm/aL0j2nX2H2J/G+6IAcWm0
LXjSVlKcaXJmmXVuUA3IJVgz39aDtX6X0TmZbyIKVnpxdMudVWkVIjdihSBmDWW09kYSIH9IfPsx
7o37gIFYGObRbT5HQoCK4dsBryod/2kjz9VcvKwwrz9YS6wjeDa8QboJGlekTZJTfiegpOCqWMLJ
```

```
ReFmWL6eqCXRxE5Jj+dEa9HlpXhKM4OdDlRnaHKvri83XyONS2bR5NAHDCm51dqpwnXbvGYYnvDb
ERXk+wTDXOJAZr79fa5PjSHiAX93R3gC9LgQ5oSF+PQYhDRf0kpy6fbzL4Nl+FtQwxrb6N124dlE
oxEIe+0Swnb5rDmSPgPZYZX6v6VFvXEJHGMid2J1oded4HxahJhC9Ig8zXJv+3x1UTsASz3phYNe
NsJCQFTlWpsGW8Ebu3ncwf5ttTpzNeG/YuX6UTmACyXw5sPAQit6tfgbBkBVhO5jI6Zf2WF3jmCZ
1PEvp/bi9ZAbHelDiN3SVIWqx+59mYYBpOVlcnrgde6JpN0DVZsl9NFec28aigYbKn81uWo5BopO
KKuewO/sBTN2lqvzZ1FGc+o+oZnSGR5iBetADBg7kkuEi6sgudzAisfVd87Bl8irrTp5QszHW9dO
/FCxP46eazl34S11lhqvo/S2pBsb9fnsI3afjUlz+6dBbkOSDXFkP8nTdmBIXs+IVbpNDn1o3eKz
89JLJ7dbEIAvUeN0qA4YhFOko3jNECpaw0VtTp4jSkGSWCmMo7IwOFPPkFrlPmRRtuf3tCA/k2Jc
rTZ9tLw6gjoCtJQiHEf+i+2G0QKnp4eJ4eJ/WmJSF4kKcnf2vFgI3R4b/6pfTzWzFG70Yl8ci5jJ
5vZB5MB4I3jPdcLTBUs8O6Dm4bbZ87z/kQVKu1vkb2B7Kx1S7F3kwGglI/aXyq4MWE0tF+B+ux0S
6EqUItaK7Bn7B8Vwj6jyty1Xw6XQrIa08Q0U3gXguwRRbkWoW8LVseZaP1qhNfCrT/CBhuD9vFUz
ediVRAA0MfYROv3VM85SUTcXCEDZ6UG1tIX7LxJLQZLszo3EM9H9GQ==</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</AAARequest>
```


Listing 6. Example real XACML AAA:Request message with the encrypted Subject element (the same encryption scenario).


```
<!-- Comment before -->
<AAA:AAARequest Id="CNLhashID#" cnl:attr1="CNL2test#" version="2.0" xmlns=""
xmlns:AAA="http://www.aaauthreach.org/ns/#AAA"
xmlns:cnl="http://www.telin.nl/ns/#cnl">
  <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
        <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">0Zj2mUBrB2gzHgXr77TIEjz1yvIpjaczXkwU
carVehi7E5woun0k5ID8TTF/JBYMaJmL4UMHs4Sd
6/VPLwGMvRPX3NAHwnXuTiWMXVNYgTzTDWWtDBkSG6xWtTTSlt2sgrMLiTWCoOvIQSE1UP/9cUYr
WYSW5ukX6j4GBpqADyg=</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">4tSi//dqNFKIizTB+CBFrquV+FJBUnCs/xEj
szVkvd1qeEq43I1jKJ3P6meMdgkwcx8AEbyXCwEz
i4YeemRs7A7xw55WwqDKuqfWwmaI38eUM+aD2A3m+Jjp6oVbGmhbp7Jk2BJeEcOJiOU6/gBTWfU8
P9XsDWft9z15t4cchjHcknj7hQqpCPcBIOCYa/jSfD6vHdmlZZxWeoGlMDxDvPk3lnQacfFo59YQ
63DKiU9vYMV9/zHWFSU2LJ4J0INMrbqv2PLVGerqp/daRCcHNt38EQIPtE6bYp/j7PRMyIJ7BA+/
ToHKgxWe7QW9tJ3GC1MPln+5wqr9KJRN4mWIfbUkgIS+f3PTcZA0uRaZydJIuSEu8NFkuzThqieQ
oQ57Jn+TbsevzXIkENaZsmK0T+U0RJGBrz1NKgdm1050i32D9f12JUkC2DbWij3RKSm+dZ5taY2c
d4cnQaLo4+m+PUNhk3FfIEdqbZ2ApCokjPx/2n7b7Y1bhlj/NXpdZdpGsIiXLJwKjKK8tBxxfTXI
HIi91GiZHqVolkoJwxGIiR01sWg/FlqEE5r+sSN0plw1hrRQFeeCOWgnOyTsVZfhzVIJL9xOt3pP
sOpRUXehu13CSEet7YMrkz8nPMYADH2mgtgin5URIjj4Da/ONpWJg3O1XtRlcYJ818MVyufhGdOM
9IUQY6AMX/LKZwI7sb8TJ9SdAPnFUbsp4XbsD7UOeaxdhOGZVuPNvww7A/2Uwvc=</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</AAA:AAARequest>
<!-- Comment after -->
```


Listing 7. Example XML Encryption of the content of the simplified AAA:Request message (using shared RSA public key for encryption of the data encryption key).

3.3 XML Encryption implementation suggestions for WSA/OCE:

XML Encryption can be used to encrypt sensitive parts of CNL Job description that provides bot BA and TA for CNL/OCE secure environment. It can also provide a method to exchange shared secrets like in case of establishing trusted relations between participating in CNL parties or VO members. Actually, XML Encryption is a part of WS-SecureConversation mechanisms.

- XML Encryption uses a security paradigm slightly different from XML Signature. XML Signature uses public key or symmetric crypto algorithm for signing quite straightforward. Major suggested/recommended XML Encryption procedure uses symmetric data encryption key (dek), often generated instantly, for encrypting data and next supplies this (instantly generated) data encryption key in the encrypted form using another symmetric key (kek) or public key of the recipient. The reason for this is that the symmetric key and data encrypted with the symmetric key have smaller size.

- Recommended in XML Encryption specification and by WS-I are the following algorithms: AES as a data encryption algorithm and key; TripleDES key wrap or RSA version 1.5 key transport for dek encryption.

- To encrypt XML document or element with Apache XML Security package for Java, it is required to provide the context document and an element to be encrypted. It important that the DOM methods used to get the element is namespace aware as both XML Encryption and XML Signature are namespace aware. This in particular related to the methods getElementsByTagNameNS(URI-NS, ElementName) which is namespace aware and getElementById(IdString) which is not namespace aware. However, the element can be identified by Id by applying ds:Reference URI transformation.

- When considering adding XML Encryption to an application, the XML document validation issues should be considered. Due to the fact that XMLEnc replaces the element or element's content, the validation against original document schema may fail. To avoid this situation, the validating parser should be XML Encryption aware , what in its own turn should carefully analysed from the security point of view, or the option of the encryption element of content should added to the XML document schema, as it is done in the SAML 2.0.

- Recommendation regarding existing XML Encryption Java packages are almost the same as for XML Digital Signature (see section 2.3 above), except that actually choice can be limited to IBM's xss4j either Apache XML Security suite, where we can recommend Apache's one.

4 Java Cryptographic Architecture (JCA)

The Java Cryptographic Architecture (JCA) provides all necessary basic cryptographic services required for PKI components management and XML security.

Java offers complete support for cryptography. For this purpose, there are several packages inside J2SE, covering all the main features of security architecture such as access controls, signatures, certificates, key pairs, key stores, and message digests.

The primary principle of JCA design is to separate cryptographic concepts from algorithmic implementations, so that different vendors can offer interoperable tools within the JCA framework.

Implementation independence is achieved using a "provider"-based architecture. The term Cryptographic Service Provider (used interchangeably with "provider" in this document) refers to a package or set of packages that implement one or more cryptographic services, such as digital signature algorithms, message digest algorithms, and key conversion services. JCA defines a series of Engine classes, where each Engine provides a cryptographic function.

JCA Engine classes

An *engine class* defines a cryptographic service in an abstract fashion (without a concrete implementation). Engine classes may have different implementation, but at the Engine API level they are all the same.

The following engine classes are defined in Java 2 SDK:

- **MessageDigest**: used to calculate the message digest (hash) of specified data.
- **Signature**: used to sign data and verify digital signatures.
- **KeyPairGenerator**: used to generate a pair of public and private keys suitable for a specified algorithm.
- **KeyFactory**: used to convert opaque cryptographic keys of type Key into key specifications (transparent representations of the underlying key material), and vice versa.
  Note.
- **CertificateFactory**: used to create public key certificates and Certificate Revocation Lists (CRLs).
- **KeyStore**: used to create and manage a keystore.A keystore is a database of keys. Private keys in a keystore have a certificate chain associated with them, which authenticates the corresponding public key. A keystore also contains certificates from trusted entities.
- **AlgorithmParameters**: used to manage the parameters for a particular algorithm, including parameter encoding and decoding.
- **AlgorithmParameterGenerator**: used to generate a set of parameters suitable for a specified algorithm.
- **SecureRandom**: used to generate random or pseudo-random numbers.
- **CertPathBuilder**: used to build certificate chains (also known as certification paths).
- **CertPathValidator**: used to validate certificate chains.
- **CertStore**: used to retrieve Certificates and CRLs from a repository.

Java Cryptographic Extensions (JCE)

All independent (third party) vendor implementations of cryptographic algorithms are called Java Cryptographic Extensions (JCE). Sun Microsystems has also provided an implementation of JCE. Whenever we use JCE, we need to configure it with JCA. For this, we need to do the following:

1. Add the address of the jar file to configure the provider (all JCE implementations are called providers) in the CLASSPATH environment variables.

2. Configure the provider in the list of your approved providers by editing `the java.security` file. This file is located in JavaHome/jre/lib/security folder. The following is the syntax to specify the priority: `security.provider.<n>=<masterClassName>.` Here, n is the priority number (1, 2, 3, etc.). MasterClassName is the name of master class to which the engine classes will call for a specific algorithm implementation. The provider's documentation will specify its master class name. For example, consider the following entries in a java.security file:

security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.rsajca.Provider
security.provider.3=com.sun.net.ssl.internal.ssl.Provider

These entries mean that the engine class will search for any algorithm implementation in the above mentioned order. It will execute the implementation found first. After these simple steps, we are all set to use JCA/JCE in our XML Encryption application.

Key management

A database called a "keystore" can be used to manage a repository of keys and certificates. A *certificate* is a digitally signed statement from one entity, saying that the public key of some other entity has a particular value. A keystore is available to applications that need it for authentication or signing purposes.

Applications can access a keystore via an implementation of the KeyStore class, which is in the java.security package. A default KeyStore implementation is provided by Sun Microsystems. It implements the keystore as a file, using a proprietary keystore type (format) named "JKS".

Applications can choose different types of keystore implementations from different providers, using the getInstance factory method supplied in the KeyStore class.

Currently, there are two command-line tools that make use of KeyStore: **keytool** and **jarsigner**, and also a GUI-based tool named **policytool**. Below are examples how to generate and save RSA key pair in keystore (changing –keyalg parameter to DSA will generate DSA keys):

1) generate RSA key pair with alias "cnl01"

```
keytool -genkey -alias cnl01 -keyalg RSA -dname "CN=AAAuthreach Security,
O=Collaboratory.nl, C=NL" -keypass xmlsecurity -keystore xmlsec/keystore1xmlsec.jks
-storepass xmlsecurity -storetype %STORETYPE%
```

2) two step procedure of extracting interesting key from the existing keystore1*, generating X.509 certificate and storing it in a new keystore2*

```
echo Copying certificate of RSA public key...
keytool -export -alias cnl01 -file xmlsec/cnl01.cer -keystore
xmlsec/keystore1xmlsec.jks -storepass xmlsecurity -storetype %STORETYPE%

keytool -import -noprompt -alias cnl01 -file xmlsec/cnl01.cer -keystore
xmlsec/keystore2xmlsec.jks -storepass xmlsecurity -storetype %STORETYPE%
```

All these functions are also available programmatically via KeyStore class that supplies well-defined interfaces to access and modify the information in a keystore.

5 References

[1] XML-Signature Syntax and Processing. – W3C REC: http://www.w3.org/TR/xmldsig-core/ Draft IETF Standard: http://www.ietf.org/rfc/rfc3275.txt

[2] XML Encryption XML Encryption Syntax and Processing - http://www.w3.org/TR/xmlenc-core/

[3] Java Cryptography Extension (JCE) Reference Guide for the Java 2 SDK, Standard Edition, v 1.4. - http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html

[4] Java Cryptography Architecture (JCA): API Specification & Reference - http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html

[5] Java Secure Socket Extension (JSSE) Reference Guide for the Java 2 SDK, Standard Edition, v 1.4. - http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html

[6] Web Services Security Framework by OASIS - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[7] WS-I Basic Security Profile Version 1.0. Working Group Draft. – May 12, 2004. - http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html

[8] Device Profile for Web Services. Microsoft Corporation. - August 2004. - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/devprof.asp