# OASIS XACML XML DSig Profile

## Working draft 0.2, 14 March 2003

**Document identifier:**

wd-aha-dsigprofile-02.sxw

**Location:**

http://www.oasis-open.org/committees/xacml/repository/

**Editor:**

Anne Anderson, Sun Microsystems <anne.anderson@sun.com>

**Contributors:**

**Abstract:**

This working draft profiles use of the W3C XML-Signature Syntax and Processing Standard in providing authentication and integrity protection for XACML schema instances.

**Status:**

This version of the specification is a working draft of the committee. As such, it is expected to change prior to adoption as an OASIS standard.

*XACML* Committee members should send comments on this specification to the xacml@lists.oasis-open.org list. Others should subscribe to and send comments to the xacml-comment@lists.oasis-open.org list. To subscribe, send an email message to xacml-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the XACML TC web page (http://www.oasis-open.org/committees/xacml/).

# Table of Contents

1Introduction........................................................3
1.1Terminology........................................................3
2XML Digital Signature Concepts........................................................5
2.1Message digest........................................................5
2.2Signature verification........................................................6
2.3Message authentication code (MAC)........................................................6
2.4Canonicalization........................................................7
2.5Signature Element format........................................................7
2.6XMLDSig Signature Types........................................................8
3XACML XMLDSig Profile........................................................10
3.1Signature type and coverage........................................................10
3.2Namespace elements in XACML data objects........................................................10

3.3Namespace elements in signatures ........ 10
3.4CanonicalizationMethod ........ 10
3.5Transform methods ........ 11
3.6Message Digest algorithms ........ 12
3.7Signature algorithms ........ 12
3.8Use of a Manifest ........ 12
3.9Signing schemas ........ 12
3.10Integrity protection for referenced external policies ........ 13
3.11Signature coverage profile ........ 14
4Examples ........ 15
4.1Basic signature for Policy1 ........ 15
4.2Basic signature for PolicySet1 and Policy1 ........ 15
4.3Enveloping signature for Manifest for PolicySet1 and Policy1 ........ 16
4.4SAML Envelope for PolicySet1 and Policy1 ........ 18
5References ........ 20

# 1Introduction

Proper use of digital signatures can provide authentication and integrity protection for XACML schema instances. **[XACML]** Sections 9.2.1 Authentication and 9.2.4 Policy integrity describe requirements and considerations for such authentication and integrity protection.

This document provides a profile for use of the W3C XML-Signature Syntax and Processing Standard in protecting OASIS eXtensible Access Control Markup Language **[XACML]** schema instances. Section 2 of this document defines terms used in the remainder of the document. Section 3 provides background information on terms and concepts associated with digital signatures and with XMLDSig in particular . Section 4 specifies guidelines for the construction of XACML schema instances that are to be signed. The guidelines in Section 4 apply to XMLDSig digital signatures as well as to other digital signature formats. Section 5 describes the formats for an XMLDSig <Reference> element that references an XACML schema instance. Only Sections 4 and 5 are normative.

This profile assumes that the XACML schema instance being signed is embedded inside of or referenced from another data object that provides information about the signer, the validity period, and other information required to make a digital signature useful: such a data object will contain or be associated with the actual digital signature that covers the XACML schema instance. This profile does not define the format for such an enclosing or referencing data object. One appropriate format that has been defined elsewhere is a **[SAML]** Assertion.

This profile SHOULD be followed when designing or using protocols that will involve the transmission of XACML Policy, PolicySet, Request, and Response instances over insecure channels. Consistent use of this profile will increase the portability and interoperability of signed data object fragments, as well as ensuring that digital signatures are being used in a way that provides the intended levels of protection.

## 1.1Terminology

*(This section is not normative.)*

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this profile are to be interpreted as described in **[RFC2119]**.

Other special terms used in this profile are defined below. When these terms occur in the profile, they are display in **bold f**ont to indicate that they are to be interpreted according to their definitions in this list.

**authentication, message** – the property that the association between an XACML **data object** instance and its **signature** can be verified.

**authentication, signer** – the property that the identity of the entity that generated a given XACML **data object** instance can be verified to be as claimed.

canonicalization – the process of producing a standard, reproducible representation for a data object.

**data object** – used in this profile to refer to a digital object that is being signed or **MAC**ed. A data object could be an XACML PolicySet, Policy, Request context, Response context, or any associated schemas. A data object is referenced inside an **Error! Bookmark not defined.**[**XMLDSIG]** <Reference> element using a URI as defined by **[RFC2396]**.

digest - see message digest.

**digital signature** – see **signature**.

**enveloped signature –** a **signature** that is included in the **data object** that is being signed**.**

**enveloping signature** – a **signature** that includes the **data object** that is being signed within its <Signature> element**.**

**detached signature** – a signature that is not attached to its associated signed **data object**. The **signature** neither envelopes nor is enveloped by the signed **data object.**

**integrity** – the property that unauthorized modifications to an XACML **data object** instance can be detected**.**

**manifest –** a structure defined by **Error! Bookmark not defined.[XMLDSIG]** that contains one or more <Reference> elements, but is not part of a <Signature> element. A <Reference> element in a <Signature> element may contain the URL and **message digest** of a **manifest.**

**message digest** – the result of applying a **one-way hash function** to a stream of bytes. Message digests are described in more detail in Section **Error! Reference source not found.**

**policy** – used in this profile to refer to instances of the XACML PolicySet and XACML Policy schemas.

**private key –** a numeric value that is used, along with the **digest** of the **data object** to be **signed,** as input to the **signature** algorithm. Each **private** key has one and only one associated **public key**. The **signer** of a **data object** must not reveal the value of the **private key** that was used to create the **signature**. In fact, it is possible to destroy the **private key** after a **signature** has been generated.

**public key –** a numeric value that is used, along with the **signature** value of a **data object**, as input to the **signature verification** algorithm. Each **public key** has one and only one associated **private key**. The **signer** of a **data object** can freely share the value of any **public key**. The **signer** *must* share the value of the **public key** with any **signature verifier** in order for **verification** to be possible. **Public keys** can be shared securely using **Public Key Certificates**.

**Public Key Certificate –** a **signed** digital structure containing the name of some entity and the value of a **public key** for which that entity owns the corresponding **private key**. A **Public Key**

**Certificate** is **signed** using a **private key** for which the **public key** can be securely obtained, often using a chain of **Public Key Certificates.**

**sign –** the process of generating a **signature**.

**signature** – a value generated by the application of a **private key** to an XACML **data object** via a cryptographic algorithm such that it has the properties of **integrity**, **message authentication**, and/or **signer authentication** (adapted from [Schneier]).

**static reference** – used in this profile to mean use of a <PolicyIdReference> or <PolicySetIdReference> where the **policy** writer wishes to refer to the snapshot of the referenced **policy** that existed at the time the referencing **policy** was written, rather than to the current contents of the referenced **policy** at the time the **policy** is to be evaluated.

t**ransform** – the process of converting an XML **data object** into a different XML **data object,** often by removing, extracting, and/or re-ordering specified elements from the original XML data object. Any **enveloped signature** must include a **transform** algorithm that will remove the **signature** value from the **data object** before the **signature** value is computed or **verified**.

**verify** – the process of checking the **signature** on a **data object** to verify that the **signature** and the **data object** are consistent.

# 2XML Digital Signature Concepts

*[This section is not normative.)*

This section explains certain concepts from the **Error! Bookmark not defined.[XMLDSIG]** specification that are needed to understand the application of a digital **signature** to an XACML **data object**.

A digital **signature** is a security mechanism that can provide some of the safeguards described in Section 9.2 of the **[XACML]** specification. In particular, it can provide a means for **authentication** of the source of an XACML **data object** and a means for ensuring the **integrity** of an XACML **data object**.

An XML Digital Signature, as used in this profile, is an XML element that contains

⑩ information about the data object that is being **signed**,

⑩the digital **signature** value itself , and

⑩information required to verify the **signature**.

In our case, the **signed** data object is an XACML schema instance or schema**.** A single **digital signature** may cover multiple data objects, and not all such data objects need to be XACML **data objects**. In our case, the **digital signature** that covers an XACML **data object** usually also covers some **data object** that contains information about the **signer**, the validity period, and so on.

A **signature** is a value computed using a cryptographic algorithm that takes as input two digital values: a stream of octets representing the value of a **message digest**, (see below) computed from the **data object** being signed, and a stream of octets representing the value of a **private key** known only to the signer of the **data object**. Associated with the **private key** is a **public key** that the signer may freely distribute to potential verifiers of **data object**s signed using the **private key**. **Public key** values are usually distributed in the form of **Public Key Certificates.**

The cryptographic algorithms used in generating a **signature** give the **signature** several useful properties:

1. it is relatively easy to compute the **signature** value,
2. it is relatively easy to compute the **message digest** from the **signature** value and the corresponding public key,
3. it is extremely difficult to determine the value of the **private key**, given the **signature** value and the **data objec**t or its **message diges**t.

Since there are several commonly used algorithms used for generating digital **signatures**, a well-known identifier for the algorithm used is included in the Signature element.

## 2.1Message digest

A **message digest** is a value computed using another cryptographic algorithm from a stream of octets representing the value of the **data object** that is being digested.  The cryptographic algorithms used in generating a **message diges**t give the **digest** value several useful properties:

1. the **digest** value is relatively short,
2. the **digest** value is relatively easy to compute,
3. any change to the digital representation of the **data object**, even by so much as one bit, will cause the **digest** computed from the **data objec**t to have a different value,
4. it is extremely difficult to generate a **data objec**t that will have a given **digest** value, so difficult that huge numbers of hours on incredibly powerful computers would be required.

Since there are several commonly used algorithms used for generating **message digests**, a well-known identifier for the algorithm used is included in the Signature element.

## 2.2Signature verification

Together, the properties of the **message digest** and the **signature** mean that the receiver of a **data object** and its associated **signature** can relatively easily verify that the **signature** goes with the provided **data object**, that the **signature** was generated using the **private key** associated with the known **public key**, and that the **data objec**t has not been modified since the **signature** was generated. If the **public key** is provided in a **Public Key Certificate**, there are similar ways to verify that the **private key** is owned by a particular identity.

The **data objec**t, the **signature**, and the **Public Key Certificate** may be stored separately and retrieved separately by the verifier. No additional protection of the channels by which these three items are transmitted are required in order to preserve the ability to perform verification of **message authentication, signer authentication**, and **integrity**.

## 2.3Message authentication code (MAC)

A **message authentication code,** or **MAC,** performs some of the functions of a **digital signature**, but does not require use of a **digital signature** algorithm: it requires only the use of a **message digest** algorithm. It requires that the generator of the **MAC** and the verifier of the **MAC** share a **secret key**.

The function for generating a **MAC** takes as input the stream of octets that represents the **data object** being **MAC**ed, and the stream of octets that represents the value of the generator's copy of the **secret key**. The **MAC** generation function temporarily appends the **secret key** octet stream to the **data object** octet stream and computes the **message digest** over the combined stream of octets. The resulting **message digest** value is the **MAC**. The **MAC** and the input **data object** (without the **secret key** value) are conveyed to some entity that needs to verify **message authentication**, **signer authentication**, and **integrity** of the **data object**.

The function for verifying a **MAC** takes as input the stream of octets that represents the **data object** that was received, the stream of octets that represents the verifier's copy of the **secret key**, and the stream of octets that represents the received **MAC** value. The **MAC** verification function temporarily appends the **secret key** octet stream to the **data object** octet stream and computes the **message digest** over the combined stream of octets. The resulting **message digest** is compared to the **MAC** received with the **data object**. If the two values match, then the **message authentication**, **signer authentication**, and **integrity** of the received **data object** have been verified. If the two values do not match, then these properties have not been verified.

The advantages of using a **MAC** are that the generation of a **message digest** is a computationally cheaper operation than the generation or verification of a **digital signature**, and that only one cryptographic algorithm – the **message digest** algorithm – needs to be supported. One disadvantage is that a new **MAC** must be generated for each sender and receiver pair, since each such pair must have a different **secret key.** This means that a MAC can not be used to protect the security of a **data object** that is to be stored in a repository shared by multiple **data object** retrievers (unless a separate **MAC** is generated and stored in the repository for each potential retriever). Another disadvantage is that the receiver is unable to pass the **MAC** and the **data object** to a third party while retaining **signer authentication** of the original signer. For these reasons, **MAC**s will be appropriate in some environments, but not in others.

## 2.4Canonicalization

Remember that even a one bit difference in the value of the **data object** will result in a different **message digest** . This means that the value of the **data object** represented as an octet stream used by the signer must be exactly identical to value of the **data object** used by the verifier. But the same XML **data object** may exist in many different forms: it may be encoded using a different character set, it may be presented in a processed form (such as a DOM or SAX representation), or certain values in the **data objec**t, such as QNames or default XML attribute values, may be represented in different ways.

In order to ensure that the digital representation of the **data object** used by the verifier is identical to the digital representation used by the signer, the signer processes the **data object** using a standard **canonicalization** method. A **canonicalization** method is a procedure that expresses all information in a **data object** in a standard, invariable way to produce a stream of octets. The **canonicalization** method used by the signer is identified in the Signature element so that the signature verifier can **canonicalize** the received **data objec**t in the same way.

## 2.5Signature Element format

The **Error! Bookmark not defined.[XMLDSIG]** Signature element has the following structure (where "?" denotes zero or one occurrence; "+" denotes one or more occurrences; and "*" denotes zero or more occurrences):.

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

The <Signature> element encompasses the digital **signature**. The <Signature> element may or may not include the **data object** being signed. This will be described below in Section **Error! Reference source not found.**.

The <SignedInfo> element is the information that is actually signed. First, the **canonicalization** algorithm specified in the <CanonicalizationMethod> element is applied to the <SignedInfo> element to produce a stream of octet values. Then the **message digest** algorithm specified in the <SignatureMethod> element is applied to that stream of octet values, producing a **message digest**. Finally, the **signature** algorithm specified in the <SignatureMethod> is applied to that **message digest**. The resulting **signature** value is placed into the <SignatureValue> element.

If a **Message Authentication Code (MAC)** is being used, then first the **canonicalization** algorithm specified in the <CanonicalizationMethod> element is applied to the <SignedInfo> element to produce a stream of octet values. Then the **MAC** algorithm is applied to the stream of octet values and the appended **secret key** value. The resulting **MAC** value is placed into the <SignatureValue> element.

The <CanonicalizationMethod> element contains the identifier of the **canonicalization** algorithm that is to be applied to the <SignedInfo> element. The result of this **canonicalization** should be a stream of octets that will be identical for a given <SignedInfo> element value, regardless of its representation.

The <SignatureMethod> element contains the identifiers of the **signature** and **message digest** algorithms (or just the **message digest** algorithm, in the case of a **MAC**). Each well-known algorithm has a well-known identifier. The <SignatureMethod> element> also contains the values of any parameters required by the chosen algorithms.

<SignedInfo> may contain any number of <Reference> elements. Each <Reference> element describes a data object to be signed using a URI. It also contains the **message digest** of the data object. Once the s**ignature** value of the <SignedInfo> element has been verified, the verifier can verify that any **data object** included in a <Reference> element in that <SignedInfo> has the same digital value as the data object that was digested originally. The verifier does this by independently computing the **message digest** value for the **data object** and comparing the resulting value with the value in the <SignedInfo>'s <Reference> element. If they match, then the **data object** has not been changed and is the **data object** that the signer intended to reference.

## 2.6XMLDSig Signature Types

**Error! Bookmark not defined.**[**XMLDSIG]** Supports four ways of using signatures.

4. **Enveloped Signature**: The <Reference> points to the **data object** that contains the <SignedInfo> element itself. In this case, the **transform** algorithm must remove the **signature** value from the **data object** before a **message digest** is calculated, since the **signature** will not be known until the **data object** is digested, but once the **signature** is inserted, the **digest** of the **data objec**t will change.

5. **Enveloping Signature**: The <Reference> points to an <Object> element that is included in the <Signature> element itself. This allows a <Signature> to be a wrapper, or envelope, around one or more signed **data object**s.

6. **Detached Signature**: The <Reference> points to a **data object** that does not contain the signature, and the signature does not contain the data object. In this case, the data object being signed may be a separate data object from the data object that contains the <Signature> element, or the data object being signed may be in the same data object as the <Signature>, but not containing or contained by the <Signature>. This way of using **signatures** allows a <Signature> element to be transported to a verifier independently from the **data object** that has been signed.

7. ***Signed Manifest**:* The <Reference> element points to a special **Error! Bookmark not defined.**[**XMLDSIG]**-defined Element called a **Manifest.** A **Manifest** is similar to a <SignedInfo> element in that it contains one or more <Reference> elements. The difference is that the rules for <SignedInfo> require that every **data object** in its <Reference> elements must be retrieved and their **message digests** verified as part of the verification of the <SignedInfo> signature. With a **Manifest**, it is up to the application to decide which **data object message digest** values must be verified, and when. This makes a **Manifest** useful when the verifier may not want to retrieve and verify every referenced **data object**.

*Note that a single <Signature> can be **enveloped**, **enveloping**, and **detached** at the same time, by including multiple <Reference> elements, each of which points to a different type of data object.*

# 3XACML XMLDSig Profile

*(This section is normative.)*

*These guidelines for using XML Signatures with XACML are intended to be consistent with Guidelines for using XML Signatures with the OASIS Security Assertion Markup Language (SAML)* **[SAMLDSig]** *wherever possible. Where the XACML recommendations must differ from the SAML recommendations below, the reasons for that difference are given. The primary source of such differences is the fact that SAML mandates use of enveloped signatures while enveloped signatures are not possible inside XACML 1.0 data objects. These mandates are not inconsistent, since the signature covering the XACML 1.0 document can be placed inside the SAML schema instance that contains the XACML document.*

## 3.1Signature type and coverage

The only XMLDSig signature type that MUST NOT be used directly with an XACML 1.0 **data object** is the **enveloped signature**. This is because there is no element defined in the XACML 1.0 schemas that can contain a **signature** that is embedded inside an XACML 1.0 schema instance.

XACML **data objects** will typically be transmitted inside an enveloping data object. The envelope in which an XACML **data object** is embedded MAY contain an **enveloped signature** that covers the XACML data object contents. As explained above, it is not currently possible to embed the **signature** over the XACML **data object** inside the XACML 1.0 **data object** itself.

When an XACML **data object** is enveloped by a SAML Assertion, then *Guidelines for using XML Signatures with the OASIS Security Assertion Markup Language (SAML)* **[SAMLDSig]** MUST be followed.

## 3.2Namespace elements in XACML data objects

Any XACML **data object** that is to be signed MUST specify all namespace elements used in the **data object**. If this is not done, then the **data object** will attract namespace definitions from ancestors of the **data object** that may differ from one envelope to another.

When **[ExclC14N]** is used as the **canonicalization** or transform method, then the namespace of XACML schemas used by elements in an XACML **data object** MUST be bound to prefixes and included in the *InclusiveNamespacesPrefixList* parameter to **[ExclC14N]**.

## 3.3Namespace elements in signatures

Since <Signature> elements are usually embedded in some protocol envelope, any <Signature> element MUST specify all namespace elements used in the <Signature> itself.  If this is not done, then the <Signature> will attract namespace definitions from ancestors of the <Signature>  that may differ from one envelope to another.

## 3.4CanonicalizationMethod

The <CanonicalizationMethod> element in a <Signature> defines how the <SignedInfo> element itself is to be **canonicalized** prior to being **digested**. The <SignedInfo> element must be converted into a specific, reproducible representation as an octet string in order for the **signature verifier** and the **signature** signer to produce the same **message digest** for the <SignedInfo> element.

**Signatures** for XACML **data objects** MUST use *Exclusive Canonicalization Version 1.0* **[ExclC14N]** (identifiers: *http://www.w3.org/2001/10/xml-exc-c14n#* and *http://www.w3.org/2001/10/xml-exc-c14n#WithComments*) as the final canonicalization algorithm if possible. If this canonicalization algorithm can not be used, then Canonical XML Version 1.0 **[InclC14N]** (identifiers: http://www.w3.org/TR/2001/REC-xml-c14n-20010315 or http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments) MUST be used.

Support for **[InclC14N]** is required in any conforming **Error! Bookmark not defined.**[**XMLDSIG]** implementation, and so use of that algorithm increases interoperability. **[ExclC14N]** however, fixes deficiencies found in **[InclC14N]**.

XACML PDPs that support **Error! Bookmark not defined.**[**XMLDSIG]** MUST be able to support both canonicalization algorithms.

See **Error! Reference source not found.** and **Error! Reference source not found.** for further considerations with respect to **canonicalization** algorithms.

## 3.5Transform methods

The <Transforms> element in a <Reference> defines **canonicalizations** and other transformations of the referenced **data object** that must be performed prior to being digested. The referenced **data object** must be converted into a specific, reproducible representation as an octet string in order for the signature verifier and the signature signer to produce the same message digest for the referenced element.

Every <Signature> for an XACML **data object** MUST use as the final transform method the same algorithm specified as the canonicalization algorithm in the <SignedInfo> element. This algorithm MUST be either **[ExclC14N]** or **[InclC14N]**, with **[ExclC14N]** preferred.

If the data object being signed is Base64-encoded, then the *Base64 Transform* (identifier: http://www.w3.org/TR/xmldsig-core/#sec-Base-64 SHOULD be used first.

If an XACML **data object** includes data elements that may be represented in more than one form (such as (TRUE, FALSE), (1,0), (true,false)), then a Transform method MUST be defined and specified for normalizing those data elements. If this is not done, the signer and the verifier may end up **digesting** different octet streams, and the **signature verification** will fail.

*[The XACML TC should specify a transform that puts all XACML-defined datatypes into their canonical form. This transform should include something like the following:*

> *The Canonical XACML Datatype Transform has the following identifier:*
>
> *urn:oasis:names:tc:xacml:1.0:transforms:canonicalDatatypeTransform*
>
> *The following canonicalizations MUST be applied to values of the corresponding datatypes, whether occurring in XML attribute values or in XACML Attributes.*
>
> 1. *Where a canonical representation for an XACML-defined datatype is defined in http://www.w3.org/2001/XMLSchema, then the value of the datatype MUST be put into the canonical form specified in http://www.w3.org/2001/XMLSchema. This includes boolean {“true”, “false”}, double, dataTime, time, date, and hexBinary (upper-case).*
> 2. *http://www.w3.org/2001/XMLSchema#anyURI - use canonical form defined in* **[RFC2396]**
> 3. *http://www.w3.org/2001/XMLSchema#base64Binary - remove all line breaks and white space. Remove all characters following the first sequence of “=” characters.*
> 4. *http://urn:oasis:names:tc:xacml:1.0:data-type:x500Name - first normalize according to* **[RFC2253]** *(leading and trailing spaces, etc.). If any RDN contains multiple attributeType*AndValue pairs, re-order the AttributeValuePairs in that RDN in ascending order when compared as octet strings (described in **[X.690]** Section 11.6 “Set-of components”).
> 5. http://urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name - normalize the domain-part of the name to lower case.
> 6. XPath expression – apply **XPath2Filt]** to put the XPath expression into canonical form. Specifying this as part of Canonical XACML DataType Transform means it does not have to be specified separately as a transform in the <Reference> element.
> 7. The definition of every new datatype added as an extension MUST include a canonical representation.
>
> *All XACML PDPs that support* **Error! Bookmark not defined.[XMLDSIG]** *must support the urn:oasis:names:tc:xacml:1.0:transforms:canonicalDatatypeTransform transform method.*
>
> *]*

See **Error! Reference source not found.** and **Error! Reference source not found.Error! Reference source not found.**for further considerations with respect to transform methods.

## 3.6Message Digest algorithms

There is only one message digest algorithm that is required for all conforming **Error! Bookmark not defined.[XMLDSIG]** implementations: SHA-1, which has identifier http://www.w3.org/2000/09/xmldsig#sha1. This algorithm MUST be used if possible for digesting the XACML data object.

XACML PDPs that support **Error! Bookmark not defined.[XMLDSIG]** MUST support SHA-1.

## 3.7Signature algorithms

There are two signature algorithms described in the **Error! Bookmark not defined.**[**XMLDSIG]** specification: DSA-SHA1, which has identifier http://www.w3.org/2000/09/xmldsig#dsa-sha1, and PKCS1 (RSA-SHA1), which has identifier http://www.w3.org/2000/09/xmldsig#rsa-sha1.

While neither of these algorithms is required for conforming **Error! Bookmark not defined.**[**XMLDSIG]** implementations, they are the algorithms most likely to be supported, and so use of one of them in signing XACML **data objects** is recommended.

XACML PDPs that support **Error! Bookmark not defined.**[**XMLDSIG]** MUST support both of these algorithms.

## 3.8Use of a Manifest

See the next two sections for a description of cases in which a **Manifest** may be appropriate.

## 3.9Signing schemas

The parsing of any XACML **data object** depends on having an accurate copy of all schemas on which the XACML **data object** depends. Note that the inclusion of a schema URI in the XACML schema instance attributes does not guarantee that an accurate copy of the schema will be used: an attacker may substitute a bogus schema that contains the same identifier as the correct schema. **Signatures** can help protect against substitution or modification of the schemas on which an XACML **data object** depends. Use of **signatures** for this purpose are described in this section.

In most cases, a **data object** signer SHOULD include a <Reference> element for each schema on which the XACML **data object** depends in the <SignedInfo> element that contains the <Reference> to or including the XACML **data object** itself.

In some cases, the **data object** signer knows that all PDPs that will evaluate a given XACML **data object** will have accurate copies of certain schemas needed to parse the **data object**, and does not want to force the PDP to verify the **message digest** for such schemas. In these cases the **data object** signer MAY omit <Reference> elements for any schema whose verification is not needed.

If the **data object** signer does not know for which schemas a PDP will have an accurate copy, then the <SignedInfo> element that contains the <Reference> to or including the XACML **data object** itself SHOULD contain a <Reference> to a <Manifest> element that, in turn, contains a <References> to each schema needed to parse the XACML **data object**. Use of a **Manifest** allows a PDP to verify the **signature** on only those schemas for which the accuracy may be in question.

## 3.10Integrity protection for referenced external policies

A **policy** signer must know the intent of the **policy** writer in determining how to generate a signature for a **policy** that contains references to other, external **data objects** via the XACML <PolicySetIdReference> and <PolicyIdReference> elements.

In many cases, a **policy** writer wishes to reference the current version of another **policy**. This can be done by using the URL of the other **policy** in a <PolicyIdReference> or <PolicySetIdReference> element. Signing the referencing **policy** does not depend on the contents of the referenced **policies**, so the current version of the referenced **policy** may be used without affecting the verification of the referencing **policy**.

In other cases, a **policy** writer wishes to reference a specific snapshot of the contents of another **policy**. We will call this a **static reference**. This can be done in either of two ways. The most straightforward way is to include the desired contents of the other **policy** as a <PolicySet> or <Policy> element. The alternative way is to use the URL of the other **policy** in a <PolicyIdReference> or <PolicySetIdReference> element, and then to sign the referencing **policy** in such a way that the **signature** includes the **message digest** of the referenced **policy** contents. This second alternative is described in the rest of this section.

The recommended way of signing a **policy** along with one or more **static references** is to use a **Manifest.** The <Manifest> element SHOULD contain a <Reference> element for each static reference in the original referencing **policy**. The <Reference> element for the original referencing **policy** MAY be in either the <Manifest> or in the <Signature> element.

The advantage of including the <Reference> for the original referencing **policy** in the **Manifest** is that the **Manifest** then becomes a package defining the **policy** and its static references. The disadvantage of including the original referencing **policy** in the **Manifest** is that verification of the <Signature> will not automatically include retrieval and verification of the original referencing **policy**, and this is almost always desired.

If the **policy** writer knows that every static reference must be retrieved as part of **policy** evaluation, or if the **policy** writer wishes to confirm that static references have not changed even if they are not used during evaluation, then a **Manifest** is not needed. In this case, the **policy** signer can include a <Reference> element for each static reference inside the <Signature> element of the original referencing **policy** itself , along with the <Reference> element for the original referencing **policy**.

## 3.11Signature coverage profile

Only the portions of a **data object** that are included in the **message digest** that is signed are actually verified when a **signature** is verified. In order to provide maximum protection for signed XACML **data objects**, this profile REQUIRES that the entire XACML **data object** be signed.

The **signature** verifier MUST verify that the entire XACML **data object** was signed by computing the **message digest** over the entire **data object**.

# 4Examples

*{This section is NOT normative.}*

## 4.1Basic signature for Policy1

This example shows a **detached signature** for a <Policy> instance named “Policy1”.  Note that the **signature verifier** must have some out-of-band means of ascertaining the identity of the signer and the validity period of this policy, since the **signature** itself does not provide this information.

```
   <Signature Id="Policy1Signature"
         xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#”/>
         <SignatureMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
         <Reference
              URI="http://www.sun.com/policies/Policy1.xml">
            <Transforms>

<Transform Algorithm=”urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </Transforms>
            <DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>?????</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>?????</SignatureValue>
      <KeyInfo>
         <KeyValue>
            <DSAKeyValue>
               <P>?????</P><Q>?????</Q><G>?????</G><Y>?????</Y>
            </DSAKeyValue>
         </KeyValue>
      </KeyInfo>
   </Signature>
```

## 4.2Basic signature for PolicySet1 and Policy1

This example shows a **detached signature** for a <PolicySet> instance named “PolicySet1” that contains a <PolicyIdReference> to a <Policy> instance named “Policy1”.  A **Manifest** is not used in this example.  Again, the **signature verifier** must have some out-of-band means of ascertaining the identity of the signer and the validity period of this policy, since the **signature** itself does not provide this information.

```
   <Signature Id="PolicySet1Policy1Signature"
        xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#”/>
         <SignatureMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
         <Reference
              URI="http://www.sun.com/policies/PolicySet1.xml">
            <Transforms>

<Transform Algorithm=”urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </Transforms>
            <DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>?????</DigestValue>
         </Reference>
         <Reference
              URI="http://www.sun.com/policies/Policy1.xml">
            <Transforms>

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </Transforms>
            <DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>?????</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>?????</SignatureValue>
      <KeyInfo>
         <KeyValue>
            <DSAKeyValue>
               <P>?????</P><Q>?????</Q><G>?????</G><Y>?????</Y>
            </DSAKeyValue>
         </KeyValue>
      </KeyInfo>
   </Signature>
```

## 4.3Enveloping signature for Manifest for PolicySet1 and Policy1

This example shows an **enveloping signature** for a **Manifest.** The **Manifest** includes <Reference> elements for a <PolicySet> instance named “PolicySet1” and for a <Policy> named “Policy1” that is referenced from “PolicySet1”. Note that the **Manifest** could have been kept as a separate XML **data object**, and not included in the <Signature> element. Once again, the **signature verifier** must have some out-of-band means of ascertaining the identity of the signer and the validity period of this policy, since the **signature** itself does not provide this information.

```
   <Signature Id="PolicySet1Policy1ManifestSignature"
        xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <SignatureMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
         <Reference
              URI="#PolicySet1Policy1Manifest">
            <Transforms>

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </Transforms>
            <DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>?????</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>?????</SignatureValue>
      <KeyInfo>
         <KeyValue>
            <DSAKeyValue>
               <P>?????</P><Q>?????</Q><G>?????</G><Y>?????</Y>
            </DSAKeyValue>
         </KeyValue>
      </KeyInfo>
      <Object>
         <Manifest Id="PolicySet1Policy1Manifest">
            <Reference
                 URI="http://www.sun.com/policies/PolicySet1.xml">
               <Transforms>

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
               </Transforms>
               <DigestMethod

Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <DigestValue>?????</DigestValue>
            </Reference>
            <Reference
                 URI="http://www.sun.com/policies/Policy1.xml">
               <Transforms>
```

```

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n "/>
               </Transforms>
               <DigestMethod

Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <DigestValue>?????</DigestValue>
            </Reference>
         </Manifest>
      </Object>
   </Signature>

```

## 4.4SAML Envelope for PolicySet1 and Policy1

This example shows how the policy used in the previous example might be enclosed in a SAML Assertion, and signed as part of the **signature** on the Assertion.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<saml:Assertion ...>
  <saml:AttributeStatement>
    <saml:Subject>
      <saml:NameIdentifier>ACMECorporateDatabase</saml:NameIdentifier>
    </saml:Subject>
    <saml:Attribute AttributeName=”urn:oasis:names:tc:xacml:1.0:policy”

AttributeNamespace=”urn:oasis:names:tc:xacml:1.0:policy”>
        <saml:AttributeValue>
            ...XACML PolicySet1 Instance goes here...
        </saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
  <ds:Signature

        Id="PolicySet1Policy1ManifestSignature"
        xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <SignatureMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
         <Reference
              URI="#PolicySet1Policy1Manifest">
            <Transforms>

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </Transforms>
            <DigestMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>?????</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>?????</SignatureValue>
      <KeyInfo>
         <KeyValue>
            <DSAKeyValue>
               <P>?????</P><Q>?????</Q><G>?????</G><Y>?????</Y>
            </DSAKeyValue>
         </KeyValue>
      </KeyInfo>
      <Object>
         <Manifest Id="PolicySet1Policy1Manifest">
            <Reference
                 URI="http://www.sun.com/policies/PolicySet1.xml">
               <Transforms>

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
               </Transforms>
               <DigestMethod

Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <DigestValue>?????</DigestValue>
            </Reference>
            <Reference
                 URI="http://www.sun.com/policies/Policy1.xml">
               <Transforms>

<Transform Algorithm="urn:oasis:names:tc:xacml:1.0:transforms:canonical
DatatypeTransform”/>
               <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#”>
                  <ec:InclusiveNamespaces PrefixList="xacml #default"
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n "/>
               </Transforms>
               <DigestMethod

Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
               <DigestValue>?????</DigestValue>
            </Reference>
         </Manifest>
      </Object>
   </ds:Signature>
</saml:Assertion>
```

The **Manifest** could also contain a <Reference> element for the XACML Policy schema.

# 5References

.

**[ExclC14N]** J. Boyer et al., *Exclusive Canonicalization Version 1.0*, http://www.w3.org/TR/xml-exc-c14n/, World Wide Web Consortium, 18 January 2002

**[InclC14N]** J. Boyer et al., Canonical XML Version 1.0, HYPERLINK "http://www.w3.org/TR/xml-c14n"http://www.w3.org/TR/xml-c14n, World Wide Web Consortium, 15 March 2001.

**[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

**[RFC2253]** M. Wahl, et al., Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names, http://www.ietf.org/rfc/rfc2253.txt, IETF RFC 2253, December 1997.

**[RFC2396]** T. Berners-Lee, et al., Uniform Resource Identifiers (URI): Generic Syntax, http://ftp://ftp.isi.edu/in-notes/rfc2396.txt, August 1998.

**[SAML]** P. Hallam-Baker, E. Maler, editors, Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML}. http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf, 31 May 2002.

**[SAMLDSig]** Scott Cantor, editor, *Guidelines for using XML Signatures with the OASIS Security Assertion Markup Language (SAML)*. http://www.oasis-open.org/committees/security/docs/draft-sstc-xmlsig-guidelines-03.pdf, Draft 03, 27 October 2002.

[Schneier] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Edition, 1996, John Wiley & Sons, Inc.

**[X.690]** ITU-T, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),* ITU-T Recommendation X.690, December 1997.

**[XACML]** S. Godik, T. Moses, *OASIS eXtensible Access Control Markup Language (XACML)*, Committee Specification Version 1.0 (Revision 1), http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-01.doc or http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-01.pdf, 12 December 2002.

**Error! Bookmark not defined.[XMLDSIG]** D. Eastlake, et al., *W3C XML-Signature Syntax and Processing*, W3C Recommendation, http://www.w3.org/TR/xmldsig-core, 12 February 2002.

**[XPath2Filt]** J. Boyer, M. Hughes, J. Reagle, editors, *XML-Signature XPath Filter 2.0*, http://www.w3.org/TR/xmldsig-filter2/, 8 November 2002.

# Appendix A.Acknowledgments

The following individuals were members of the committee during the development of this specification:

-

In addition, the following people made contributions to this specification:

- .

# Appendix B.Revision History

| Rev | Date | By Whom | What |
|---|---|---|---|
| p-03 | 2003-01-15 | Anne Anderson | Initial version |
| wd-02 | 2003-03-14 | Anne Anderson | Make consistent with SAML Digital Signature Guidelines, Incorporate MAC usage, SAML assertion example, various edits based on comments received. |
| | | | |

# Appendix C.Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.