# XML Signature 1.0 Requirements Specification

## 1. Scope

### 1.1 Overview

This component will provide the ability to digitally sign and verify XML documents.
The WSE 3.0 does not currently provide support for signing and encrypting SOAP messages on the .Net compact framework. This component is part of a set of components that will provide the minimum functionality to help fill this gap.
This component currently supports the SHA1 algorithm for computing the digest value. A plug-in is required for this component that will allow for the SHA256 algorithm to be supported as well. Basically, an implementation of the `IDigester` interface is required, that will implement the SHA256 algorithm.

### 1.2 Logic Requirements

#### *1.2.1 Standard*

This component will follow the specifications defined by the *http://www.w3.org/TR/xmldsig-core/* standard to provide the functionality described in the following sections.

#### *1.2.2 Signing an XML document*

This component will generate the `Signature` element, and the sub elements and attributes as described below.

##### 1.2.2.1 References

- URI references to the XML elements to be signed will be specified and included in `Reference` elements.
- Each reference element can optionally be transformed to the canonical form. This information will be included in the `Transforms` element.
- A hashing algorithm should be specified for calculating the digest. The used algorithm will be included in the `DigestMethod` element. Support for SHA 1 algorithm is required.
  Support for SHA256 algorithm is also required. Since the .Net Compact Framework does not implement this algorithm, this plug-in will have to follow the algorithm description to provide an implementation. (http://en.wikipedia.org/wiki/SHA256 and http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf).
  An example implementation can be found here: http://www.mperfect.net/cfAes/
- The digest value should be computed using the specified hashing algorithm and included in the `DigestValue` element, base64 encoded.

##### 1.2.2.2 Signed info

- All the `Reference` elements will be grouped inside a `SignedInfo` element.
- A `CanonicalizationMethod` element will be added to the `SignedInfo` element, to specify the algorithm used to canonize the included references.
- The algorithm used to calculate the hash of the signature and the algorithm used to encrypt the hash will be specified in the `SignatureMethod` element. Support for the RSA and DSA asymmetric encryption algorithms is required.

##### 1.2.2.3 Signing

Calculate the digest of the `Reference` elements using the specified hashing algorithm, encrypt the digest using a client private key provided to the component and put the signature value in a `SignatureValue` element, base64 encoded.

#### 1.2.2.4 Key information

Key information will be included in a `KeyInfo` element so that the signature can be decrypted and validated. The key that will be used for decryption is the public key included in the sender X.509 certificate. This component will support adding custom key identification using the XML namespace facility.

### *1.2.3 Verifying the signature of `SignedInfo` element*

Recalculate the digest of the `Reference` elements (using the digest algorithm specified in the `DigestMethod` elements) and the digest of the `SignedInfo` element (using the digest algorithm specified in the `SignatureMethod` element) and use the public verification key to decrypt and verify that the value of the `SignatureValue` element is correct for the digest of the `SignedInfo` element.

### *1.2.4 Canonical XML*

Prior to generating the digest value, the `Reference` elements that will be signed need to be in canonical form. Also, when creating the `Signature` elements they will also be in canonical form. Canonical specifications can be found here: http://www.w3.org/TR/xml-c14n.
A tutorial can also be found here: http://www-128.ibm.com/developerworks/xml/library/x-c14n/

## 1.3 Required Algorithms

- The component will support the SHA 1 algorithm for calculating the digest.
- The component will support the SHA 256 algorithm for calculating the digest.
- The component will support the RSA and DSA asymmetric encryption algorithms for signing and verifying the signature.

## 1.4 Example of the Software Usage

An application that acts as a WSE client wants to digitally sign outgoing soap messages and verify the signature of incoming soap messages.

Details and examples can be found here: http://www.xml.com/pub/a/2001/08/08/xmldsig.html

## 1.5 Future Component Direction

Support for all elements defined by the standard could be added.

# 2. Interface Requirements

### *2.1.1 Graphical User Interface Requirements*

None.

### *2.1.2 External Interfaces*

None.

### *2.1.3 Environment Requirements*

- Development language: C#
- Compile target: .Net Compact Framework 2.0

### *2.1.4 Package Structure*

TopCoder.Security.Crytography.Mobile

## 3. Software Requirements

### 3.1 Administration Requirements

#### *3.1.1 What elements of the application need to be configurable?*

None.

### 3.2 Technical Constraints

#### *3.2.1 Are there particular frameworks or standards that are required?*

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

#### *3.2.2 TopCoder Software Component Dependencies:*

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

- Canonical XML

#### *3.2.3 Third Party Component, Library, or Product Dependencies:*

None.

#### *3.2.4 QA Environment:*

.Net Compact Framework 2.0
Mobile 5.0

### 3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

### 3.4 Required Documentation

#### *3.4.1 Design Documentation*

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

#### *3.4.2 Help / User Documentation*

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.