DSS Extension for Local Signature Computation Version 1.0

Working Draft 01

10 January 2012

Specification URIs:

This Version:

http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0-wd01.html (Authoritative)
http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0-wd01.pdf
http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0-wd01.xml

Previous Version:

http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0.html
http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0.pdf
http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0.xml

Latest Version:

http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0.html (Authoritative)
http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0.pdf
http://docs.oasis-open.org/dss-x/profiles/localsig/localsig-v1.0.xml

Technical Committee:

OASIS Digital Signature Services eXtended (DSS-X) TC

Chairs:

Juan Carlos Cruellas, UPC-DAC <cruellas@ac.upc.edu>
Stefan Drees, Individual <stefan@drees.name>

Editor:

Ernst Jan van Nigtevecht, Sonnenglanz Consulting BV <ejvn@sonnenglanz.net>

Author:

Frank Cornelis, FedICT <frank.cornelis@fedict.be>

Related Work:

This specification is related to:

- oasis-dss-core-spec-v1.0-os

and may be combined with other existing profiles, such as

- oasis-dss-profiles-AdES-v1.0-os
- oasis-dssx-1.0-profiles-vr-cs01

for example.

Declared XML Namespaces:

urn:oasis:names:tc:dss-x:1.0:profiles:localsig:schema#

Abstract:

This document defines a protocol and processing profiles of [DSSCore] , which allows to create document signatures using local signature computation. Finally, it defines transport and security bindings for the protocols.

Status:

This Working Draft (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or approved as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document Approval Process begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

Table of Contents

Appendixes

1. Introduction

The signing protocols of the OASIS DSS specification [DSSCore] mainly focus on centralized key management systems. Such architecture makes sense for situations where the connecting clients don't own tokens with signing capabilities themselves. However, large-scale signing token deployments (e.g. national eID cards) reduce the need for a centralized key management system. In such scenarios it is still interesting to keep a centralized system in place for several reasons:

- Despite the fact that every person owns a token with signing capability, he/she might not have the appropriate software installed on the system for the creation of electronic signatures. It might be easier to maintain a lightweight applet solution, instead of a full blown token middleware that has to be installed on every participating client's system. The diversity among the client platforms is also easier to manage from a centralized platform instead of by distributing token middleware to all participating clients. Furthermore, managing the configuration of the signature policy to be used for creation and validation of signatures within a certain business context might be easier using a centralized platform.

- When transforming a paper-world business workflow to a digital equivalent that includes the creation and/or validation of signatures, it might be interesting to offer the sub-process of creating/validating electronic signatures as an online service. Given the technicality of signature creation and validation, a clean separation of concerns in the service architecture is desired.

- From a technical point of view it might be easier to maintain different DSS servers each specializing in handling a specific token type. E.g. tokens per vendor, or per country.

So the role of the centralized system shifts from key management to providing a platform that manages the technicalities of signing documents using client tokens. Such digital signature service systems require a new set of protocol messages for the creation of signatures where signature computation is accomplished via local tokens.

1.1. Terminology

The key words *MUST* , *MUST NOT* , *REQUIRED* , *SHALL* , *SHALL NOT* , *SHOULD* , *SHOULD NOT* , *RECOMMENDED* , *MAY* , and *OPTIONAL* are to be interpreted as described in [RFC 2119] .

These keywords are capitalized when used to unambiguously specify requirements over protocol features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

This specification uses the following typographical conventions in text: `<ns:Element>` , `Attribute` , **Datatype** , `OtherCode` .

1.2. Normative References

[DSSAsync] A. Kuehne et al., Asynchronous Processing Abstract Profile of the OASIS Digital Signature Services Version 1.0 , OASIS, April 2007 *http://docs.oasis-open.org/dss/v1.0/oasis-dss-profiles-asynchronous_processing-spec-v1.0-os.pdf* *[http://docs.oasis-open.org/dss/v1.0/oasis-dss-profiles-asynchronous_processing-spec-v1.0-os.pdf]*

[DSSCore] S. Drees et al., Digital Signature Service Core Protocols and Elements , OASIS, April 2007 *http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf* *[http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf]*

[DSSVer] D. Hühnlein et al., Profile for Comprehensive Multi-Signature Verification Reports Version 1.0 , OASIS, November 2010 *http://docs.oasis-open.org/dss-x/profiles/verificationreport/oasis-dssx-1.0-profiles-vr-cs01.pdf* *[http://docs.oasis-open.org/dss-x/profiles/verificationreport/oasis-dssx-1.0-profiles-vr-cs01.pdf]*

[Excl-C14N] J. Boyer et al., Exclusive XML Canonicalization Version 1.0 , World Wide Web Consortium, July 2002 *http://www.w3.org/TR/xml-exc-c14n/ [http://www.w3.org/TR/xml-exc-c14n/]*

[HTML401] D. Raggett et al., HTML 4.01 Specification , World Wide Web Consortium, December 1999 *http://www.w3.org/TR/html4 [http://www.w3.org/TR/html4]*

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

[RFC 2616] R. Fielding et al., Hypertext Transfer Protocol - HTTP/1.1. , *http://www.ietf.org/rfc/rfc2616.txt* IETF (Internet Engineering Task Force) RFC 2616, June 1999.

[SAMLCore] S. Cantor et al., Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 , OASIS, March 2005 *http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf [http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf]*

[WS-SecConv] A. Nadalin et al., WS-SecureConversation 1.4 , OASIS, February 2009 *http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html [http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html]*

[WS-Trust] A. Nadalin et al., WS-Trust 1.3 , OASIS, March 2007 *http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html [http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html]*

[XHTML] XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) , World Wide Web Consortium Recommendation, August 2002 *http://www.w3.org/TR/xhtml1/ [http://www.w3.org/TR/xhtml1/]*

[XMLSig] D. Eastlake et al., XML-Signature Syntax and Processing , W3C Recommendation, June 2008 *http://www.w3.org/TR/xmldsig-core/*

[XML-ns] T. Bray, D. Hollander, A. Layman, Namespaces in XML , W3C Recommendation, January 1999 *http://www.w3.org/TR/1999/REC-xml-names-19990114 [http://www.w3.org/TR/1999/REC-xml-names-19990114]*

1.3. Namespaces

The structures described in this specification are contained in the schema file which is part of Section A.1, “Schema” . All schema listings in the current document are excerpts from the schema file. This schema is associated with the following XML namespace:

```
urn:oasis:names:tc:dss-x:1.0:profiles:localsig:schema#
```

Conventional XML namespace prefixes are used in this document:

- The prefix `ds:` stands for the W3C XML Signature namespace [XMLSig]
- The prefix `dss:` stands for the DSS core namespace [DSSCore]
- The prefix `async:` stands for the DSS Asynchronous Processing Abstract Profile namespace [DSSAsync]
- The prefix `vr:` stands for the Profile for Comprehensive Multi-Signature Verification Reports namespace [DSSVer]
- The prefix `wst:` stands for the WS-Trust namespace [WS-Trust]

Applications MAY use different namespaces, and MAY use whatever namespace defaulting/scoping conventions they desire, as long as they are compliant with the Namespace in XML specification [XML-ns] .

2. DSS Protocol Messages

The DSS signing protocol messages defined in [DSSCore] inherently assume that all security aspects are covered by the transport binding and appropriate security binding. Unfortunately some transport bindings require that the security is also handled at the protocol message level. The DSS protocol messages defined in this section leave room for such protocol message level security.

DSS protocol messages can be generated and exchanged using a variety of protocols. The binding specifications under Section 3, "DSS Protocol Bindings" describes specific means of transporting protocol messages using existing, widely deployed transport protocols.

The following schema fragment defines the XML namespaces and other header information for the protocol schema:

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="urn:oasis:names:tc:dss-x:1.0:profiles:localsig:schema#"
 elementFormDefault="qualified" attributeFormDefault="unqualified"
 xmlns:tns="urn:oasis:names:tc:dss-x:1.0:profiles:localsig:schema#"
 xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <import namespace="urn:oasis:names:tc:dss:1.0:core:schema"
  schemaLocation="http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-schema-v1.0-os.xsd

 <import namespace="http://www.w3.org/2000/09/xmldsig#"
  schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd" />

 ...
</schema>
```

2.1. Signature Request

This section defines the protocol message `<SignRequest>` used for signature creation requests.

The `<SignRequest>` message SHOULD be signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message.

The `<SignRequest>` message has complex type `SignRequestType` and contains the following attributes and elements:

`ID` [Required]
: An identifier for the request. It is of type `xs:ID` .

`IssueInstant` [Required]
: The time instant of issue of the request. The time value is of type `xs:dateTime` and is encoded in UTC.

`Destination` [Optional]
: A URI reference indicating the address to which this request has been sent. This is useful to prevent malicious forwarding of requests to unintended recipients, a protection that is required by some protocol bindings. If it is present, the actual recipient MUST check that the URI reference identifies the location at which the message was received. If it does not, the request MUST be discarded. Some protocol bindings may require the use of this attribute.

`Consent` [Optional]
: Indicates whether or not (and under what conditions) consent has been obtained from a principal in the sending of this request. See Section 8.4 of [SAMLCore] for some URI references that MAY be used as the value of the `Consent` attribute and their associated descriptions.

`Issuer` [Optional]
: Identifies the entity that generated the request message.

`ProtocolBinding` [Optional]
: A URI reference that identifies a DSS protocol binding to be used when returning the `<SignResponse>` message. See Section 3, "DSS Protocol Bindings" for more information about protocol bindings and URI references defined for them.

`ProviderName` [Optional]
: Specifies the human-readable name of the requester for use by the presenter's user agent or the DSS.

`<dss:SignRequest>` [Required]
: The actual OASIS DSS Core [DSSCore] signing request message.

`<ds:Signature>` [Optional]
: An XML Signature [XMLSig] that authenticates the requester and provides message integrity.

`<Extensions>` [Optional]
: This extension point contains optional protocol message extension elements that are agreed on between the communicating parties. No extension schema is required in order to make use of this extension point, and even if one is provided, the lax validation setting does not impose a requirement for the extension to be valid. DSS extension elements MUST be namespace-qualified in a non-DSS-defined namespace.

Depending on the requirements of particular protocols or profiles, a DSS requester may often need to authenticate itself, and message integrity may often be required. Authentication and message integrity MAY be provided by mechanisms provided by the protocol binding (see [DSSCore]). The DSS request MAY be signed, which provides both authentication of the requester and message integrity.

If such a signature is used, then the `<ds:Signature>` element MUST be present, and the DSS responder MUST verify that the signature is valid (that is, that the message has not been tampered with) in accordance with [XMLSig] . If it is invalid, then the responder MUST NOT rely on the contents of the request and SHOULD respond with an error. If it is valid, then the responder SHOULD evaluate the signature to determine the identity and appropriateness of the signer and may continue to process the request or respond with an error (if the request is invalid for some other reason).

The schema for this element is listed below:

```
<element name="SignRequest" type="tns:SignRequestType" />

<complexType name="SignRequestType">
 <sequence>
  <element ref="dss:SignRequest" />
  <element ref="ds:Signature" minOccurs="0" />
  <element name="Extensions" type="dss:AnyType" minOccurs="0" />
 </sequence>
 <attribute name="ID" type="xs:ID" use="required" />
 <attribute name="IssueInstant" type="xs:dateTime" use="required" />
 <attribute name="Destination" type="xs:anyURI" use="optional" />
 <attribute name="Consent" type="xs:anyURI" use="optional" />
 <attribute name="Issuer" type="xs:anyURI" use="optional" />
 <attribute name="ProtocolBinding" type="xs:anyURI" use="optional" />
```

```
  <attribute name="ProviderName" type="xs:string" use="optional" />
 </complexType>
```

2.2. Signature Response

This section defines the protocol message <SignResponse> used for signature creation responses.

The <SignResponse> message SHOULD be signed or otherwise authenticated and integrity protected by the protocol binding used to deliver the message.

The <SignResponse> message has complex type SignResponseType and contains the following attributes and elements:

ID [Required]
: An identifier for the response. It is of type xs:ID .

InResponseTo [Optional]
: A reference to the identifier of the request to which the response corresponds, if any. If the response is not generated in response to a request, or if the ID attribute value of a request cannot be determined (for example, the request is malformed), then this attribute MUST NOT be present. Otherwise, it MUST be present and its value MUST match the value of the corresponding request's ID attribute.

IssueInstant [Required]
: The time instant of issue of the response. The time value is of type xs:dateTime and is encoded in UTC.

Destination [Optional]
: A URI reference indicating the address to which this response has been sent. This is useful to prevent malicious forwarding of requests to unintended recipients, a protection that is required by some protocol bindings. If it is present, the actual recipient MUST check that the URI reference identifies the location at which the message was received. If it does not, the request MUST be discarded. Some protocol bindings may require the use of this attribute.

Consent [Optional]
: Indicates whether or not (and under what conditions) consent has been obtained from a principal in the sending of this response. See Section 8.4 of [SAMLCore] for some URI references that MAY be used as the value of the Consent attribute and their associated descriptions.

Issuer [Optional]
: Identifies the entity that generated the response message.

<dss:Result> [Required]
: The DSS Core [DSSCore] result.

<dss:SignResponse> [Optional]
: The actual DSS Core [DSSCore] signing response message.

<ds:Signature> [Optional]
: An XML Signature [XMLSig] that authenticates the requester and provides message integrity.

<Extensions> [Optional]
: This extension point contains optional protocol message extension elements that are agreed on between the communicating parties. No extension schema is required in order to make use of this extension point, and even if one is provided, the lax validation setting does not impose a requirement for the extension to be valid. DSS extension elements MUST be namespace-qualified in a non-DSS-defined namespace.

Depending on the requirements of particular protocols or profiles, a DSS responder may often need to authenticate itself, and message integrity may often be required. Authentication and message integrity

MAY be provided by mechanisms provided by the protocol binding (see [DSSCore]). The DSS response MAY be signed, which provides both authentication of the responder and message integrity.

If such a signature is used, then the <ds:Signature> element MUST be present, and the DSS requester receiving the response MUST verify that the signature is valid (that is, that the message has not been tampered with) in accordance with [XMLSig] . If it is invalid, then the requester MUST NOT rely on the contents of the response and SHOULD treat it as an error. If it is valid, then the requester SHOULD evaluate the signature to determine the identity and appropriateness of the signer and may continue to process the response as it deems appropriate.

The schema for this element is listed below:

```
<element name="SignResponse" type="tns:SignResponseType" />

<complexType name="SignResponseType">
 <sequence>
  <element ref="dss:Result" />
  <element ref="dss:SignResponse" minOccurs="0" />
  <element ref="ds:Signature" minOccurs="0" />
  <element name="Extensions" type="dss:AnyType" minOccurs="0" />
 </sequence>
 <attribute name="ID" type="xs:ID" use="required" />
 <attribute name="InResponseTo" type="xs:NCName" use="optional" />
 <attribute name="IssueInstant" type="xs:dateTime" use="required" />
 <attribute name="Destination" type="xs:anyURI" use="optional" />
 <attribute name="Consent" type="xs:anyURI" use="optional" />
 <attribute name="Issuer" type="xs:anyURI" use="optional" />
</complexType>
```

3. DSS Protocol Bindings

The following sections define the protocol bindings. DSS bindings are categorized under either transport bindings or security bindings as defined under section 6 of [DSSCore] .

3.1. HTTP POST Transport Binding

The HTTP POST binding defines a mechanism by which DSS protocol messages may be transmitted within the base64-encoded content of an HTML form control.

The reference URI for this binding is

`urn:oasis:names:tc:dss-x:1.0:profiles:localsig:bindings:HTTP-POST`

3.1.1. Overview

The HTTP POST binding is intended for cases in which the DSS requester and responder need to communicate using an HTTP user agent (as defined in HTTP 1.1 [RFC 2616]) as an intermediary. This may be necessary, for example, if the communicating parties do not share a direct path of communication. It may also be needed if the responder requires an interaction with the user agent in order to fulfill the request, such as when the user agent must authenticate to it.

3.1.2. RelayState

RelayState is a mechanism for preserving and conveying state information. If a DSS request message is accompanied by RelayState data, then the DSS responder MUST return its DSS protocol response using a binding that also supports a RelayState mechanism, and it MUST place the exact data it received with the request into the corresponding RelayState parameter in the response.

3.1.3. Message Encoding

Messages are encoded for use with this binding by encoding the XML into an HTML form control and are transmitted using the HTTP POST method. A DSS protocol message is form-encoded by applying the base-64 encoding rules to the XML representation of the message and placing the result in a hidden form control within a form as defined by [HTML401] Section 17. The HTML document MUST adhere to the XHTML 1.0 specification [XHTML] to ease parsing. The base64-encoded value MAY be line-wrapped at a reasonable length in accordance with common practice.

If the message is a DSS signing request, then the form control MUST be named `SignRequest` . If the message is a DSS signing response, then the form control MUST be named `SignResponse` . Any additional form controls or presentation MAY be included but MUST NOT be required in order for the recipient to process the message.

If a "RelayState" value is to accompany the DSS protocol message, it MUST be placed in an additional hidden form control named `RelayState` within the same form with the DSS message.

The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using this binding to which the DSS message is to be delivered. The `method` attribute MUST be "POST".

Any technique supported by the user agent MAY be used to cause the submission of the form, and any form content necessary to support this MAY be included, such as submit controls and client-side scripting commands. However, the recipient MUST be able to process the message regardless for the mechanism by which the form submission is initiated.

Note that any form control values included MUST be transformed so as to be safe to include in the XHTML document. This includes transforming characters such as quotes into HTML entities, etc.

3.1.4. HTTP and Caching Considerations

HTTP proxies and the user agent intermediary should not cache DSS protocol messages. To ensure this, the following rules SHOULD be followed. When returning DSS protocol messages using HTTP 1.1, HTTP responders SHOULD:

- Include a `Cache-Control` header field set to " `no-cache, no-store` ".
- Include a `Pragma` header field set to " `no-cache` ".

There are no other restrictions on the use of HTTP headers.

3.2. Message Security Binding

Certain transport bindings do not yield the required protocol security properties when the DSS protocol message passes through an intermediary. The security binding described in this section can ensure these required security properties.

The XML signature [XMLSig] elements defined within the `<SignRequest>` and `<SignResponse>` protocol messages can provide authentication, and integrity to the Section 3.1, “HTTP POST Transport Binding” , or others.

The XML Signature specification [XMLSig] calls out a general XML syntax for signing data with flexibility and many choices. This security binding details constraints on these facilities so that DSS processors do not have to deal with the full generality of XML Signature processing. This usage makes specific use of the **xs:ID** -typed attributes present on the root elements to which signatures can apply, specifically the `ID` attribute on the `<SignRequest>` request and `<SignResponse>` response elements. These attributes are collectively referred to in this section as the identifier attributes.

Note that this security binding only applies to the use of the `<ds:Signature>` elements found directly within `<SignRequest>` requests, and `<SignResponse>` responses.

3.2.1. Signing Formats and Algorithms

XML Signature has three ways of relating a signature to a document: enveloping, enveloped, and detached.

The DSS `<SignRequest>` and `<SignResponse>` protocol messages MUST use enveloped signatures when signing the protocol messages. DSS processors SHOULD support the use of RSA signing and verification for public key operations in accordance with the algorithm identified by `http://www.w3.org/2000/09/xmldsig#rsa-sha1` . DSS processors SHOULD support the use of HMAC signing and verification for secrets in accordance with the algorithm identified by `http://www.w3.org/2000/09/xmldsig#hmac-sha1` .

3.2.2. References

DSS protocol messages MUST supply a value for the `ID` attribute on the root element of the protocol message being signed. The protocol message's root element may or may not be the root element of the actual XML document containing the protocol message (e.g., it might be contained within a SOAP envelope).

Signatures MUST contain a single `<ds:Reference>` containing a same-document reference to the `ID` attribute value of the root element of the protocol message being signed. For example, if the `ID` attribute value is "foo", then the `URI` attribute in the `<ds:Reference>` element MUST be "#foo".

3.2.3. Canonicalization Method

DSS implementations SHOULD use Exclusive Canonicalization [Excl-C14N] , without comments, both in the `<ds:CanonicalizationMethod>` element of `<ds:SignedInfo>` , and as a `<ds:Transform>`

algorithm. Use of Exclusive Canonicalization ensures that signatures created over DSS protocol messages embedded in an XML context can be verified independent of that context.

3.2.4. Transforms

Signatures in DSS protocol messages SHOULD NOT contain transforms other than the enveloped signature transform (with the identifier `http://www.w3.org/2000/09/xmldsig#enveloped-signature`) or the exclusive canonicalization transforms (with the identifier `http://www.w3.org/2001/10/xml-exc-c14n#`).

Verifiers of signatures MAY reject signatures that contain other transform algorithms as invalid. If they do not, verifiers MUST ensure that no content of the DSS protocol message is excluded from the signature. This can be accomplished by establishing out-of-band agreement as to what transforms are acceptable, or by applying the transforms manually to the content and reverifying the result as consisting of the same DSS protocol message.

3.2.5. KeyInfo

XML Signature defines usage of the `<ds:KeyInfo>` element. DSS does not require the use of `<ds:KeyInfo>` , nor does it impose any restrictions on its use. Therefore, `<ds:KeyInfo>` MAY be absent.

3.3. Secure Conversation Security Binding

The secret for HMAC signing as defined within section Section 3.2, “Message Security Binding” can be acquired via different out-of-bound mechanisms. This security binding allows for piggybacking of WS-SecureConversation [WS-SecConv] messages over DSS protocol messages for the establishment of a security context and corresponding session key derivation.

A `<wst:RequestSecurityToken>` secure conversation request message can be put as optional input in a DSS protocol request message. A `<wst:RequestSecurityTokenResponseCollection>` secure conversation response message can be put as optional output in a DSS protocol response message.

For example, the following secure conversation request message can be put as optional input within a DSS protocol request message.

```
<wst:RequestSecurityToken
 xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
 <wst:TokenType>
  http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
 </wst:TokenType>
 <wst:RequestType>
  http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
 </wst:RequestType>
 <wst:Entropy>
  <wst:BinarySecret
   Type="http://docs.oasis-open.org/ws-sx/ws-trust/200512/Nonce">
  ...
  </wst:BinarySecret>
 </wst:Entropy>
 <wst:KeySize>256</wst:KeySize>
</wst:RequestSecurityToken>
```

The following example security conversation response message can be delivered by the DSS as optional output within a DSS protocol response message.

```
<wst:RequestSecurityTokenResponseCollection
 xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
 xmlns:wsc="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512"
 xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
 <wst:RequestSecurityTokenResponse>
  <wst:TokenType>
   http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
  </wst:TokenType>
  <wst:RequestedSecurityToken>
   <wsc:SecurityContextToken>
    <wsc:Identifier>token-id</wsc:Identifier>
   </wsc:SecurityContextToken>
  </wst:RequestedSecurityToken>
  <wst:RequestedUnattachedReference>
   <wsse:SecurityTokenReference>
    <wsse:Reference
    ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct"
    URI="token-id" />
   </wsse:SecurityTokenReference>
  </wst:RequestedUnattachedReference>
  <wst:RequestedProofToken>
   <wst:ComputedKey>
    http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1
   </wst:ComputedKey>
  </wst:RequestedProofToken>
  <wst:Entropy>
   <wst:BinarySecret
    Type="http://docs.oasis-open.org/ws-sx/ws-trust/200512/Nonce">
   ...
   </wst:BinarySecret>
  </wst:Entropy>
  <wst:KeySize>256</wst:KeySize>
  <wst:Lifetime>
   <wsu:Created>2012-01-13T01:00:00.000Z</wsu:Created>
   <wsu:Expires>2012-01-13T02:00:00.000Z</wsu:Expires>
  </wst:Lifetime>
 </wst:RequestSecurityTokenResponse>
</wst:RequestSecurityTokenResponseCollection>
```

The established security context can be used within the Section 3.2, “Message Security Binding” for signing the protocol messages as follows:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
 <ds:SignedInfo>
  <ds:CanonicalizationMethod
   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  <ds:SignatureMethod
   Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
  <ds:Reference URI="#SignRequestId">
   <ds:Transforms>
    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   </ds:Transforms>
   <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   <ds:DigestValue>...</ds:DigestValue>
```

```
  </ds:Reference>
 </ds:SignedInfo>
 <ds:SignatureValue>...</ds:SignatureValue>
 <ds:KeyInfo>
  <wsse:SecurityTokenReference>
   <wsse:Reference
    ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct"
    URI="token-id" />
  </wsse:SecurityTokenReference>
 </ds:KeyInfo>
</ds:Signature>
```

The secure conversation context can be cancelled by piggybacking the following message.

```
<wst:RequestSecurityToken
 xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
 xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
 wsu:Id="RST">
 <wst:RequestType>
  http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel
 </wst:RequestType>
 <wst:CancelTarget>
  <wsse:SecurityTokenReference>
   <wsse:Reference
    ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct"
    URI="token-id" />
  </wsse:SecurityTokenReference>
 </wst:CancelTarget>
 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
   <ds:CanonicalizationMethod
    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
   <ds:SignatureMethod
    Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
   <ds:Reference URI="#RST">
    <ds:Transforms>
     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <ds:DigestValue>...</ds:DigestValue>
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>...</ds:SignatureValue>
  <ds:KeyInfo>
   <wsse:SecurityTokenReference>
    <wsse:Reference
     ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct"
     URI="token-id" />
   </wsse:SecurityTokenReference>
  </ds:KeyInfo>
 </ds:Signature>
</wst:RequestSecurityToken>
```

In this WS-Trust token request message, the required proof of possession XML signature of the key associated with the security context has been embedded within the <wst:RequestSecurityToken> element.

The DSS can answer with the following message:

```
<wst:RequestSecurityTokenResponseCollection
 xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
 <wst:RequestSecurityTokenResponse>
  <wst:RequestedTokenCancelled />
 </wst:RequestSecurityTokenResponse>
</wst:RequestSecurityTokenResponseCollection>
```

4. Local Signature Computation Profile

When using the <SignRequest> and <SignResponse> messages defined in Section 2, “DSS Protocol Messages” the following URI SHALL be added to the <dss:AdditionalProfile> element within the embedded <dss:SignRequest> element.

urn:oasis:names:tc:dss-x:1.0:profiles:localsig

A protocol transport binding as defined under Section 3, “DSS Protocol Bindings” SHALL be used. The Section 3.1, “HTTP POST Transport Binding” SHOULD be used in combination with Section 3.2, “Message Security Binding” , together with the TLS Security Bindings as defined under section 6.3 of [DSSCore] .

The relying party SHOULD include the optional input <dss:Language> element as defined in [DSSCore] section 2.8.3 to indicate the preferred localization to be used by the DSS server.

In case the end-user cancelled the signing operation, the service returns in <SignResponse> a <dss:ResultMajor> of RequesterError and a <dss:ResultMinor> of

urn:oasis:names:tc:dss-x:1.0:profiles:localsig:resultminor:user-cancelled

In case the DSS service detects a problem with the client runtime environment, the service returns in <SignResponse> a <dss:ResultMajor> of RequesterError and a <dss:ResultMinor> of

urn:oasis:names:tc:dss-x:1.0:profiles:localsig:resultminor:client-runtime

4.1. Claimed Identity

The relying party can include the optional <dss:ClaimedIdentity> element as defined in [DSSCore] section 2.8.2 to indicate the identity of the client who is making the request. The information provided by <dss:ClaimedIdentity> can be used to further personalize the interface presented to the end-user by the DSS server.

The <dss:SupportingInfo> element MAY contain a <SecurityToken> element. This element is of complex type SecurityTokenType and contains the following attributes and elements:

TokenType [Required]
: The URI reference indicating the type of the embedded token. Token type URIs are typically defined in token profiles such as those in the OASIS WSS TC.

In case the DSS server detects a problem with the claimed identity, the service returns in <SignResponse> a <dss:ResultMajor> of RequesterError and a <dss:ResultMinor> of

urn:oasis:names:tc:dss-x:1.0:profiles:localsig:resultminor:claimed-identity

The schema for this element is listed below:

```
<element name="SecurityToken" type="tns:SecurityTokenType" />

<complexType name="SecurityTokenType">
 <complexContent>
  <extension base="dss:AnyType">
   <attribute name="TokenType" type="xs:anyURI" use="required" />
  </extension>
 </complexContent>
</complexType>
```

4.2. Signer Identity

Because of the local signature computation the relying party can not always determine in advance which digital identity will be used by the client to sign the document. Different strategies are possible to eventually determine this digital identity. The relying party can for example verify the signed document via a DSS verifying protocol as defined under section 4 of [DSSCore] for this. Via the elements defined within this section we give the relying party additional means to determine the signatory's digital identity.

We allow usage of the `<dss:ReturnSignerIdentity>` optional input and corresponding `<dss:SignerIdentity>` optional output as defined under section 4.5.7 of [DSSCore] within the context of signature creation. Notice that these elements were originally defined within the context of signature verification.

The presence of the `<dss:ReturnSignerIdentity>` optional input instructs the DSS server to return a `<dss:SignerIdentity>` optional output as part of the signature creation process.

5. Artifact Profile

Passing large documents for signing via a user agent may result in a bad end-user experience due to document transfer delays. Hence the need for a profile that uses a SOAP web service for passing the documents between relying party and DSS. This profile is based on the DSS Asynchronous Processing Abstract Profile [DSSAsync] .

For the establishment of a shared secret the Section 3.3, “Secure Conversation Security Binding” can be used. When using this security binding, the lifecycle of the document that is stored in the DSS temporary document repository SHOULD correspond with the lifecycle of the secure conversation context.

5.1. Prepare for signing

This section describes the request and response messages to prepare for a signing operation.

The actual document signing ceremony is initated via the DSS protocol messages defined in Section 2, “DSS Protocol Messages” and involves the user agent.

5.1.1. SignRequest

The `<dss:SignRequest>` message is used to transfer the to be signed document to the DSS server.

Add an `<dss:AdditionalProfile>` element containing the following URI to use this profile.

`urn:oasis:names:tc:dss-x:1.0:profiles:localsig:asynchronousprocessing`

5.1.2. SignResponse

This profile uses the following `<dss:ResultMajor>` code as defined in [DSSAsync] to indicate that the operation did not finish yet.

`urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending`

This profile uses the `<async:ResponseID>` optional output element as defined in [DSSAsync] as reference to the document that has been stored in the DSS server.

5.2. Signing

This builds on Section 4, “Local Signature Computation Profile” .

The `<SignRequest>` and `<SignResponse>` messages as defined in Section 2, “DSS Protocol Messages” SHALL include the document reference. This document reference is used to correlate the document transmitted via Section 5.1, “Prepare for signing” and the current user agent session.

5.2.1. SignRequest

Add an `<dss:AdditionalProfile>` element to the embedded `<dss:SignRequest>` containing the following URI to use this profile.

`urn:oasis:names:tc:dss-x:1.0:profiles:localsig:asynchronousprocessing`

This profile uses the optional input element `<async:ResponseID>` as defined in [DSSAsync] as reference to the document.

5.2.2. SignResponse

If the server returns the `<dss:ResultMajor>` code

`urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending`

then the document signature was created successfully. In this case the `<dss:SignResponse>` within response message contains the optional output element `<async:ResponseID>` as defined in [DSSAsync] .

5.3. Finalize signing

This section describes the request and response messages to finalize a signing operation.

5.3.1. SignRequest

Add an `<dss:AdditionalProfile>` element containing the following URI to use this profile.

`urn:oasis:names:tc:dss-x:1.0:profiles:localsig:asynchronousprocessing`

This profile uses the optional input element `<async:ResponseID>` as defined in [DSSAsync] as reference to the document.

5.3.2. SignResponse

The response message contains the signed document.

6. Original Document Verification Profile

When the signing messages between the DSS and the relying party are passed via a user agent, various additional attack vectors are possible. This profile can be used to provide the relying party additional means to verify whether no MITM attack occurred.

We profile the DSS Verifying Protocol as defined in section 4 of [DSSCore] .

This profile can be used in combination with other profiles like [DSSVer] .

6.1. VerifyRequest

Add an `<dss:AdditionalProfile>` element containing the following URI to use this profile.

`urn:oasis:names:tc:dss-x:1.0:profiles:localsig:original-document`

The `<dss:Document>` element MUST have an `ID` attribute to uniquely identify the document within a particular request message.

The `<OriginalDocument>` optional input element contains an original document. Multiple `<OriginalDocument>` elements are allowed. The `<OriginalDocument>` element is of type `OriginalDocumentType` and contains the following attributes and elements.

`WhichDocument` [Required]
: A reference to the `<dss:Document>` element within the `<dss:InputDocuments>` element that contains the corresponding signed document. It is of type `xs:IDREF` .

`<dss:Document>` [Required]
: This element contains the original document.

The schema for this element is listed below:

```
<element name="OriginalDocument" type="tns:OriginalDocumentType" />

<complexType name="OriginalDocumentType">
 <sequence>
  <element ref="dss:Document" />
 </sequence>
 <attribute name="WhichDocument" type="xs:IDREF" use="required" />
</complexType>
```

6.2. VerifyResponse

In case the service detected a mismatch between a signed document and the corresponding original document, the service returns a `<dss:ResultMajor>` of `RequesterError` and a `<dss:ResultMinor>` of:

`urn:oasis:names:tc:dss-x:1.0:profiles:localsig:resultminor:changed-document`

In case the `<dss:VerifyResponse>` contains a verification report as defined in [DSSVer] the above defined URI MAY be used as value for `<dss:ResultMinor>` within a `<vr:IndividualReport>` element.

7. Security Considerations

Before deployment, each combination of profile, transport binding, and security binding SHOULD be analyzed for vulnerability in the context of the specific protocol exchange and the deployment environment. Below we illustrate some of the security concerns that often come up with protocols of this type, but we stress that this is not an exhaustive list of concerns.

If a message defined under Section 2, “DSS Protocol Messages” is signed, the `Destination` XML attribute in the root DSS protocol mesage element MUST contain the URL to which the sender has instructed the user agent to deliver the message. The recipient MUST then verify that the value matches the location at which the message has been received.

As the DSS protocol defined in this document is similar to the SAML protocol most of the security considerations defined in [SAMLCore] also apply to the DSS protocol.

As opposed to signatures created in the context of entity authentication, creation of document signatures using the DSS protocol yields additional attack vectors as the client may benefit to greater extent from manipulation of the to be signed document being transferred between relying party and DSS server. If the end user signs a different document as assumed by the relying party, the business impact could be huge.

In the context of document signatures it is of eminent importance to even properly secure the DSS protocol request message that is transferred from relying party to the DSS server via the intermediate user agent. This in order to avoid undetectable document manipulations by for example the end user. The Section 3.3, “Secure Conversation Security Binding” can ensure this.

Appendix A. Normative Annex

A.1. Schema

The XML Schema for Local Signature Computation is based on the SAML protocol messages [SAMLCore] .

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="urn:oasis:names:tc:dss-x:1.0:profiles:localsig:schema#"
 elementFormDefault="qualified" attributeFormDefault="unqualified"
 xmlns:tns="urn:oasis:names:tc:dss-x:1.0:profiles:localsig:schema#"
 xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <import namespace="urn:oasis:names:tc:dss:1.0:core:schema"
  schemaLocation="http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-schema-v1.0-os.xsd

 <import namespace="http://www.w3.org/2000/09/xmldsig#"
  schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd" />

 <element name="SignRequest" type="tns:SignRequestType" />

 <complexType name="SignRequestType">
  <sequence>
   <element ref="dss:SignRequest" />
   <element ref="ds:Signature" minOccurs="0" />
   <element name="Extensions" type="dss:AnyType" minOccurs="0" />
  </sequence>
  <attribute name="ID" type="xs:ID" use="required" />
  <attribute name="IssueInstant" type="xs:dateTime" use="required" />
  <attribute name="Destination" type="xs:anyURI" use="optional" />
  <attribute name="Consent" type="xs:anyURI" use="optional" />
  <attribute name="Issuer" type="xs:anyURI" use="optional" />
  <attribute name="ProtocolBinding" type="xs:anyURI" use="optional" />
  <attribute name="ProviderName" type="xs:string" use="optional" />
 </complexType>

 <element name="SignResponse" type="tns:SignResponseType" />

 <complexType name="SignResponseType">
  <sequence>
   <element ref="dss:Result" />
   <element ref="dss:SignResponse" minOccurs="0" />
   <element ref="ds:Signature" minOccurs="0" />
   <element name="Extensions" type="dss:AnyType" minOccurs="0" />
  </sequence>
  <attribute name="ID" type="xs:ID" use="required" />
  <attribute name="InResponseTo" type="xs:NCName" use="optional" />
  <attribute name="IssueInstant" type="xs:dateTime" use="required" />
  <attribute name="Destination" type="xs:anyURI" use="optional" />
  <attribute name="Consent" type="xs:anyURI" use="optional" />
  <attribute name="Issuer" type="xs:anyURI" use="optional" />
 </complexType>
```

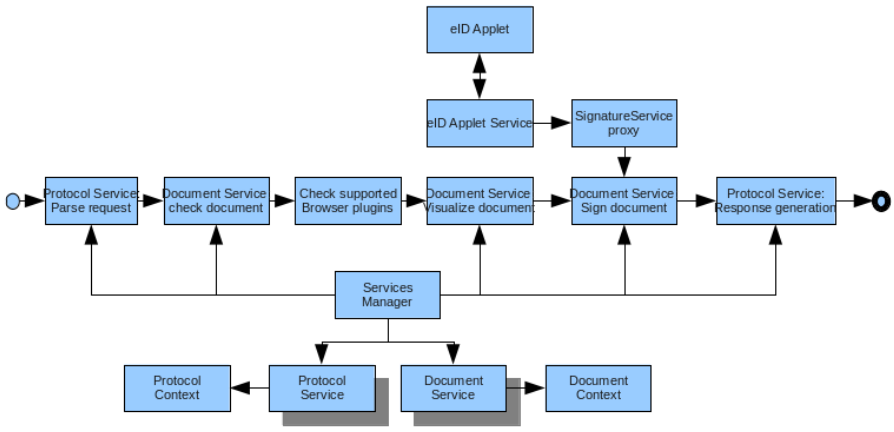
```
 <element name="OriginalDocument" type="tns:OriginalDocumentType" />

 <complexType name="OriginalDocumentType">
  <sequence>
   <element ref="dss:Document" />
  </sequence>
  <attribute name="WhichDocument" type="xs:IDREF" use="required" />
 </complexType>

 <element name="SecurityToken" type="tns:SecurityTokenType" />

 <complexType name="SecurityTokenType">
  <complexContent>
   <extension base="dss:AnyType">
    <attribute name="TokenType" type="xs:anyURI" use="required" />
   </extension>
  </complexContent>
 </complexType>
</schema>
```

Appendix B. Non-normative Annex (Non-Normative)

B.1. Sample Application

Figure B.1, “eID DSS Signature Pipeline” shows the design of a digital signature service that uses the Belgian eID card as client signing token.

Figure B.1. eID DSS Signature Pipeline

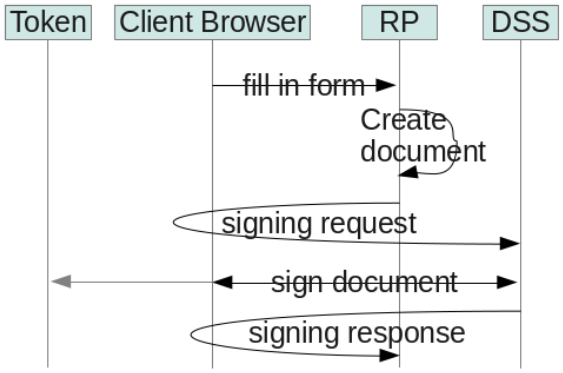


An end-user enters the DSS signature pipeline via some protocol. First of all the appropriate protocol service parses the request. At this step the mime type of the incoming document is determined. Via the mime type the appropriate document service can be selected. The document service will first check the incoming document (syntax, ...). Next the web browser capabilities are being queried in order for the document service to be able to correctly visualize the received document. After the user's consent the document service will orchestrate the document signing process using a web browser Java applet component. Finally the signed document is returned via the protocol service that also handled the incoming protocol request.

The advantage of such a generic signature pipeline architecture is that one can easily add new document formats by providing a new document service implementation. Because the protocol handling is also isolated in protocol services, one can also easily add new DSS protocols to the platform. Another advantage of such a signature pipeline is that every Relying Party that uses the platform is guaranteed that the user followed a certain signature ceremony and is fully aware of the content of the signed document. This guarantee can be interesting from a legal point of view.

A sample protocol flow is shown in Figure B.2, “Sequence diagram of a simple protocol flow” .

Figure B.2. Sequence diagram of a simple protocol flow



Here the client navigates via a web browser to the web application of the relying party. As part of the business work flow, the client fills in a web form. The relying party's web application converts the received form data into a document that needs to be signed by the client. Now the relying party's web application redirects the client web browser to the DSS web application. The DSS web application takes care of the signing ceremony using Java applet technology to connect to the client's token. Finally the DSS web application redirects the client's web browser back to the relying party. The relying party can now further process the signed document as part of the implemented business work flow.

In such scenarios it is difficult to use the existing OASIS DSS protocol messages as is, because the OASIS DSS protocol does not provide the security mechanisms required to secure the communication between relying parties and the DSS in the context of web browsers. Various MITM attacks are possible at different points during the signature ceremony. Similar to the OASIS SAML Browser POST profile, we need to define additional wrapper messages to be able to guarantee secure transportation of the DSS requests and responses via web browsers.

A disadvantage of the simple protocol shown is that the entire document is being transferred between relying party and DSS (and back) using the client's web browser. Given the upload limitation of most client's internet connection, this might result in a bad end-user experience when trying to sign a large document. So additionally we should define some form of artifact binding. Here the relying party sends the to be signed document via a SOAP DSS web service to the DSS. The DSS stores the document in some temporary document repository. The relying party receives back a document identifier which it passes as parameter when redirecting the client's web browser towards the DSS. At the end of the protocol flow, the relying party can fetch the signed document from the DSS web service using the document identifier.

Appendix C. Acknowledgements (Non-Normative)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

- Alice
- Bob
- Malory

Appendix D. Revision History (Non-Normative)

Revision 0.0	November 3, 2011	OASIS
Initial document obtained from OASIS		
Revision 0.1	November 7, 2011	E.J. Van Nigtevecht
Added chapters and a short description.		
Revision 0.2	December 18, 2011	F. Cornelis
Added initial protocol messages and profiles.		
Revision 0.2.1	December 23, 2011	F. Cornelis
user-cancelled resultminor code.		
Revision 0.2.2	January 2, 2012	F. Cornelis
Protocol binding improvements and ClaimedIdentity.		
Revision 0.2.3	January 3, 2012	F. Cornelis
XML schema fixes and security binding.		
Revision 0.2.4	January 4, 2012	F. Cornelis
Signer identity and secure conversation security binding.		
Revision 0.2.5	January 10, 2012	F. Cornelis
Secure conversation context cancellation.		