DSS Extension for Local Signature Computation Version 1.0

Working Draft 01

8 November 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>

Authors:

Frank Cornelis, FedICT <frank.cornelis@fedict.be>
Ernst Jan van Nigtevecht, Sonnenglanz Consulting BV <ejvn@sonnenglanz.net>

Related Work:

This specification is related to:

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

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], 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

Table of Contents

Appendixes

1. Introduction

The OASIS Digital Signature Services specification [DSSCore] standardizes a protocol by which (i) a client can send documents (or document hashes) to a server and receive back a signature on the documents, or by which (ii) a client can send documents (or document hashes) and a signature to a server, and receive back an answer on whether the signature verifies the documents.

These operations could be useful in a variety of contexts, for example they could allow clients to access a single corporate key for signing press releases, with centralized access control, auditing, and archiving of signature requests. They could also allow clients to create and verify signatures without needing complex client software and configuration.

This profile extends the OASIS DSS protocol such that a (secure) signature creation device (an SSCD or SCD), under the direct control of the user, can be used. The (secure) signature creation device is not part of, nor located at, the server that implements the DSS protocol.

The (secure) signature creation device may have limited software and performance capabilities and hence may be supported by a OASIS DSS compliant service to handle the complexities of the signature creation and document manipulation.

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. Abbreviations

- *SCD*: Signature Creation Device. A device that is capable of creating a digital signature using a private key that is stored in the device.
- *SSCD*: Secure Signature Creation Device. A device that is capable of creating a digital signature using a private key that is stored in the device; the private key cannot be copied from the device.
- *LSCD*: Local Signature Creation Device. A (secure) signature creation device that is owned and possessed by an enduser.
- *RSCD*: Remote Signature Creation Device. A (secure) signature creation device that is owned, but not possessed, by an enduser; nonetheless, the device is under the control of the enduser.
- *Client*: A requester of a particular resource or service that is provided by a server.
- *Server*: A provider of a resource or service that is used by a client.
- *RP*: Relying Party. An entity on the Internet that uses an identity provider to authenticate a user.

1.3. 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*

[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*

[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*

[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/*

[HTML401] D. Raggett et al., HTML 4.01 Specification, World Wide Web Consortium, December 1999, *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*

[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*

[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*

[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*

1.4. Non-Normative References

[ECC] CEN CEN-TS 15480 / CEN/TC 224 - Personal identification, electronic signature and cards and their related systems and operations

[M-COMM] ETSI Mobile Commerce (M-COMM); Mobile Signatures; Business and Functional RequirementsETSI Technical Report 102 203 V1.1.1, May 2003

1.5. 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 OASIS DSS core namespace [DSSCore]
- The prefix `async:` stands for the OASIS 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].

1.6. Requirements (Non-Normative)

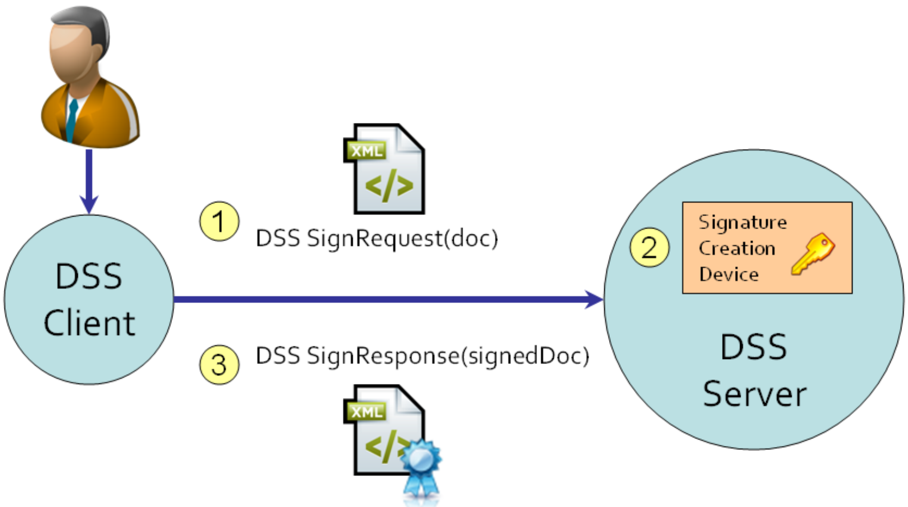
This section lists the requirements for the local signature computation. The overall goal is to extend the OASIS Digital Signature Service (DSS) protocol such that a signature can be created by means of an (secure) signature creation device under the direct control of an enduser.

- It shall be possible to use a (secure) signature creation device (using protocols such as [ISO/IEC 7816], [ISO/IEC 24727] and [CEN 15480]) at a different location from the OASIS DSS server.
- It shall be possible to specify a hash algorithm to be used in the signature creation process.
- It shall be possible to obtain the hash value for a given input document (and document type) and given hash algorithm.
- It shall be possible for a given PKCS#1 signature together with a given hash algorithm and given input document (and document type) to obtain the signed document, using the given signature.

1.7. Design Rationale (Non-Normative)

The DSS protocol assumes a client-server relationship. The client initiates a SignRequest (1) and the server signs the document (2). The resulting document is sent back to the client in the SignResponse (3). This is shown in the figure below.

Figure 1. A client and server that implement the OASIS DSS protocol.



Note that the signature creation device (SCD) is (by default) part of the server that implements the OASIS DSS protocol.

Such an architecture is applicable in case the endusers do not own a signature creation device (SCD). However, large-scale signing token deployments increase the use of signature creation devices that are owned and/or possessed by an enduser. Examples are:

- National eID cards or European Citizen Card [ECC], containing a secure signature creation device (SSCD).
- Mobile devices, where the SIM card can be used as a secure signature creation device (SSCD) [M-COMM].

In such scenarios it is still interesting to keep a OASIS DSS 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 solution, for instance by means of an applet, 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 service instead of distributing token middleware to all participating client systems. Furthermore, managing the configuration of the signature policy to be used for creation and validation of signatures within a certain context might be easier using a centralized service.

- When transforming a paper-world business workflow to a digital equivalent that includes the creation and/or validation of signatures, a sub-process for creating and validating electronic signatures as a service which can be easily integrated with a business application.

- From a technical point of view it might be easier to maintain different OASIS DSS services, each specialized in handling a specific signature and token types. E.g. tokens per vendor, or per country.

This profile extends the OASIS DSS protocol such that an enduser can present its own (secure) signature creation device; the device itself is not located at the server that implements the DSS protocol.

Although the (secure) signature creation device is under the control of the user, the location of the device can be local or remote. The following terminology is used:

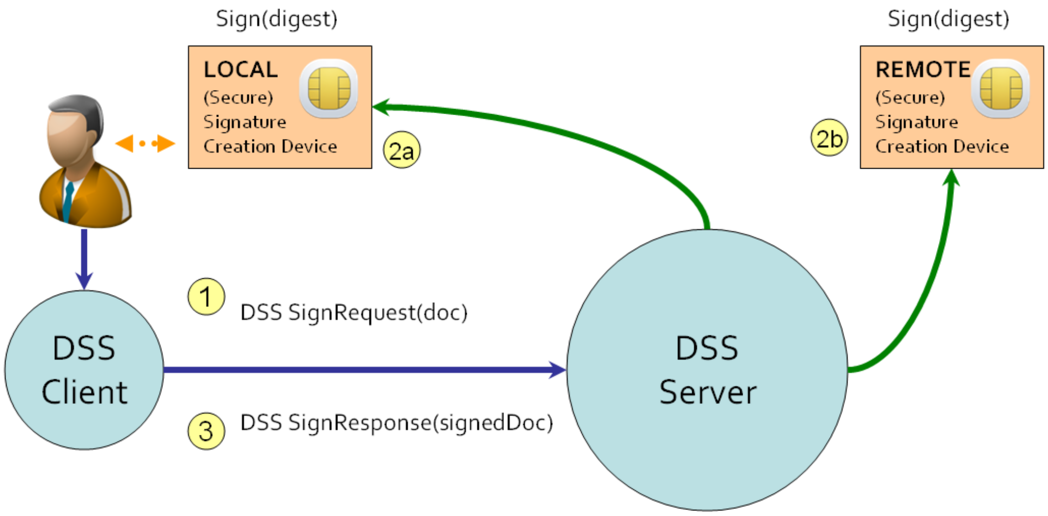
- if the (secure) signature creation device is owned and possessed by the enduser, it is referred to as a *LOCAL* (secure) signature creation device, abbreviated to *LSCD*;

- if the (secure) signature creation device is owned by, but not possessed by the enduser (but still under the direct control of the enduser) it is referred to as a *REMOTE* (secure) signature creation device, abbreviated to *RSCD*.

The LSCD is in the neighbourhood of the enduser, which is not the case for the RSCD. (Note that from the viewpoint of the DSS server, both LSCD and RSCD have to be treated as remote devices.)

The following diagram visualizes the relationship between the LSCD or RSCD and the DSS server. Note that the connection between the LSCD resp. RSCD and the DSS Server still has to be defined; the logical relationship is depicted in green.

Figure 2. Local and remote device for signature creation.

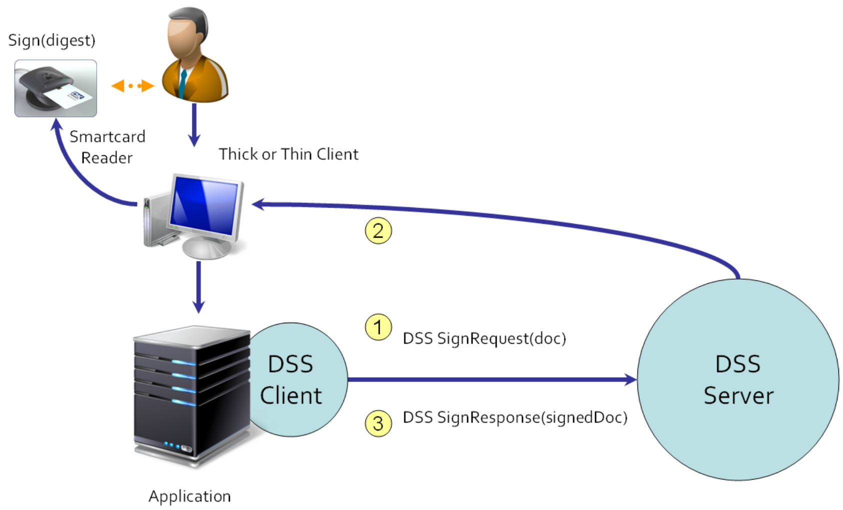


The LSCD or RSCD may have limited software and performance capabilities and hence may be supported by a OASIS DSS compliant service to handle the complexities of the signature creation and document manipulation. The LSCD or RSCD will serve a request to sign a given digest. It is assumed that the interface to the actual (S)SCD is accessed through one of the possible standards, such as the APDU (ISO 7816) or the IFD-Client (ISO/IEC 24727 / CEN 15480) standard. This shows that the profile should not depend on the actual interface of the (S)SCD. It is assumed that there will be some middleware that abstracts from the vendor-specific implementation of the (S)SCD.

Because a connection is required between the LSCD or RSCD a new profile is required to define the access protocol. Unfortunately, the use of the LSCD and RSCD depends on the actual use case. Three examples are presented, restricted to LSCD's.

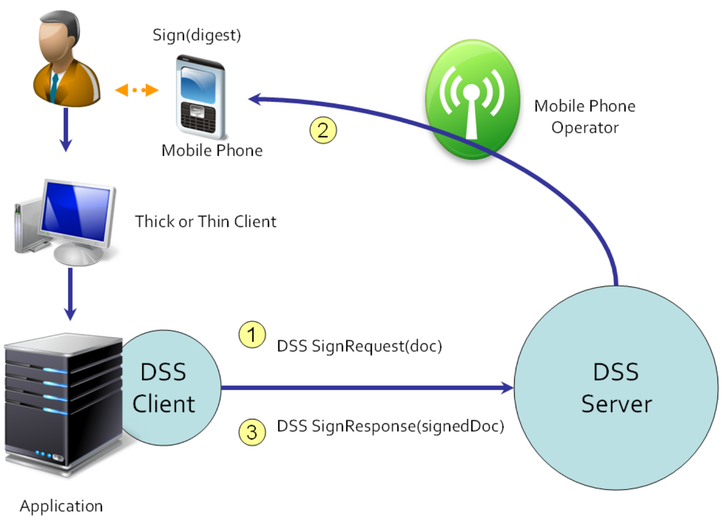
1. This example assumes a thick or thin client platform (not to be confused with the DSS client); the signature creation device is a smartcard. A webbrowser is used to access an application that also implements a DSS client.

Figure 3. A smartcard use case.



2. This example assumes the use of a mobile phone that contains a (secure) signature creation device. The mobile phone is connected to the mobile operator infrastructure. A webbrowser is used to access an aplication that also implements a DSS client. The DSS server connects to the mobile phone. This use case resembles the ETSI standard for Mobile Commerce (M-COMM) or Mobile Signature Service [M-COMM].

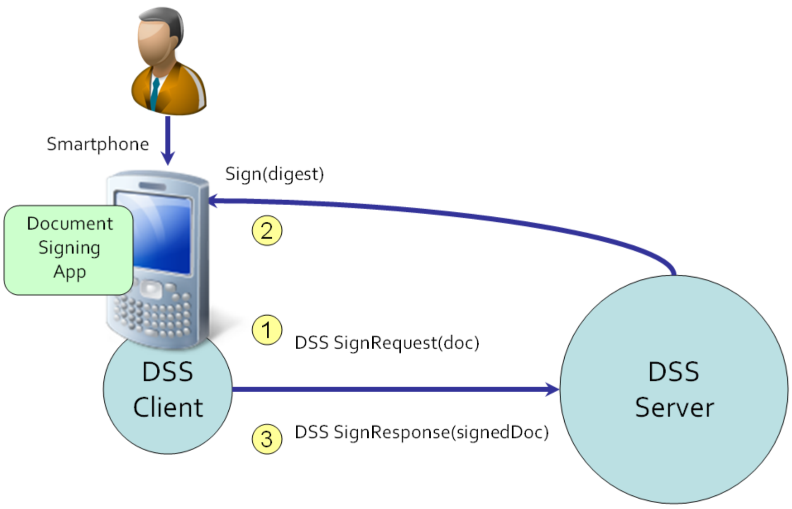
Figure 4. A mobile phone use case.



3. This example assumes the use of a smartphone or tablet that contains a (secure) signature creation device. The smartphone or tablet contains an app that implements an application to sign documents,

although the actual document signature handling functionality is delegated to a DSS server; the app implements a DSS client.

Figure 5. A smartphone use case.



1.7.1. Variant 1

The following mechanisms are introduced to enable the use of an LSCD by a DSS client:

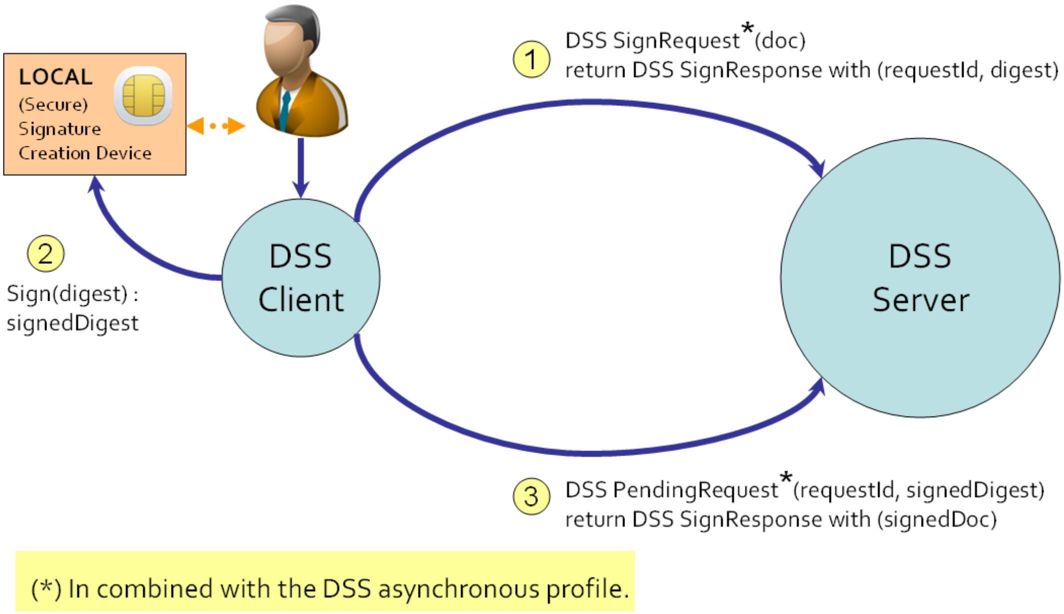
1. Asynchronous processing, as defined in [DSSAsync], of the SignRequest and SignResponse. Note: the following restriction is relaxed for the use of the `<dss:DocumentHash>` element (see next bullet): *If the server returns the `<ResultMajor>` code 'Pending' the contents of the `<OptionalOutputs>` element children other than `<async:ResponseID>` are undefined.*

2. An OptionalOutput `<dss:DocumentHash>` element in the SignResponse. After the DSS server has received a SignRequest it calculates the digest of the document. The SignResponse contains a `<dss:ResultMajor>` value 'Pending' as well as the `<async:ResponseID>` and the `<dss:DocumentHash>` values.

3. An OptionalInput `<dss:SignatureObject>` element in the PendingRequest. After the DSS client has received the digest value as a result from the SignRequest, it creates a signature. The resulting signature is included in the PendingRequest by means of the `<dss:SignatureObject>` element (the `<async:ResponseID>` is provided as well as). The DSS server will perform the necessary operations to incorporate the signed digest into the corresponding document. The resulting signed document is returned in the SignResponse with a `<dss:ResultMajor>` value 'Success'.

The use of the LSCD by the DSS client is depicted below. Although it is shown that the DSS client accesses the LSCD, other solutions are possible, for instance if application implements a DSS client and an IFD-Client (ISO/IEC 24727 / CEN 15480).

Figure 6. An LSCD used by the DSS client.



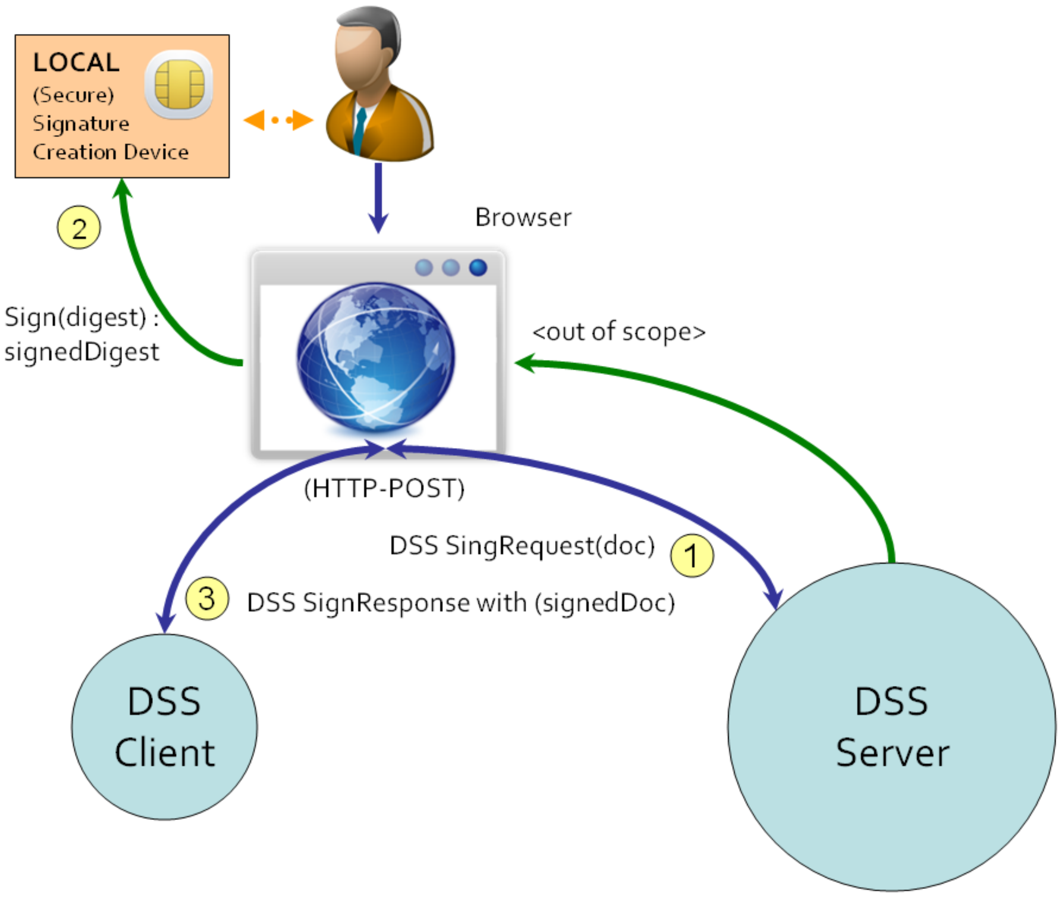
1.7.2. Variant 2a

The following mechanisms are introduced to enable the use of an LSCD by a DSS server:

1. An existing webbrowser session.

An enduser visits a webapplication and selects a document that has to be signed (the webapplication implements a DSS client). The webapplication responds to the webbrowser with an HTTP-POST request that contains the SignRequest. The HTTP-POST request contains code that will redirect the request to the DSS server, thereby creating a session between the webbrowser and the DSS server (1). The DSS server presents a screen in which the enduser can sign the document using the LSCD (2). When the enduser has finished, the DSS server will respond to the webbrowser with a HTTP-POST request that contains the SignResponse. The HTTP-POST request contains code that will redirect the request to the DSS client, thereby returning the SignResponse to the DSS client (3).

Figure 7. An LSCD used by the DSS server



1.7.3. Variant 2b

The following mechanisms are introduced to enable the use of an LSCD by a DSS server:

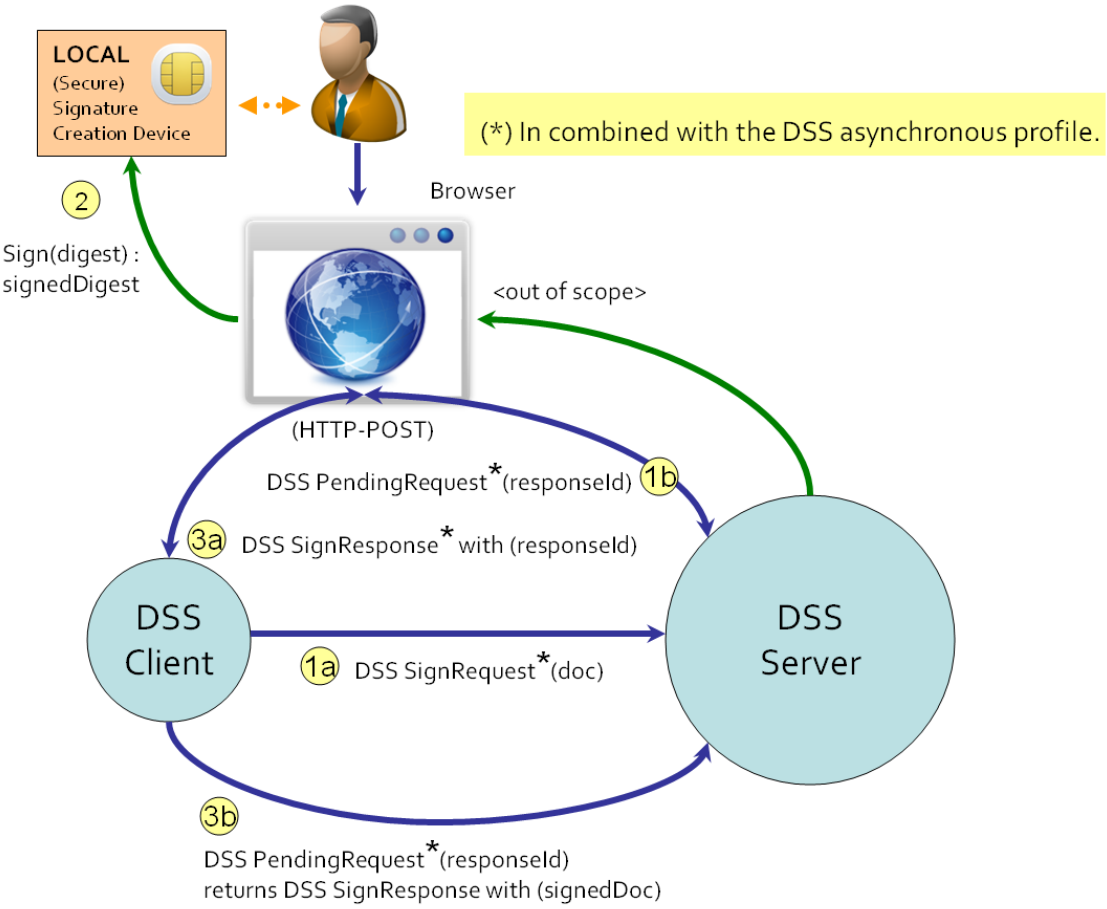
1. An existing webbrowser session (similar to variant 2a).

2. Secure conversation, to secure the different asynchronous requests and responses between the DSS client and DSS server.

3. Asynchronous processing profile, as defined in [DSSAsync]. Note: the following restriction is relaxed for the use of the `<wst:SecurityToken>` element (required for the secure conversation): *If the server returns the `<ResultMajor>` code 'Pending' the contents of the `<OptionalOutputs>` element children other than `<async:ResponseID>` are undefined.*

This variant is similar to the previous one, except that the document is sent asynchronous. The DSS client first sends the document to the DSS server (1a) after which an HTTP-POST request is sent (via the webbrowser to the DSS server) with a PendingRequest (1b). The DSS server presents a screen in which the enduser can sign the document using the LSCD (2). When the enduser has finished, the DSS server responds to the webbrowser with a HTTP-POST request that contains a SignResponse (together with a request ID related to the resulting document). The HTTP-POST request contains code that will redirect the request to the DSS client, thereby returning the SignResponse to the DSS client (3a). The DSS client retrieves the resulting document form the DSS server with a PendingRequest (3b).

Figure 8. An LSCD used by the DSS server



2. Profile Features

2.1. Identifier

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

2.2. Scope

This profile extends the OASIS DSS signing functionalitiy, as defined in [DSSCore], such that an enduser can present it's own (secure) signature creation device.

This profile is restricted to LSCD's that can be accessed by means of a webbrowser.

The following restrictions apply:

- the enduser will access an application by means of a webbrowser;
- the webbrowser may be capable of accessing the LSCD through some interface (such as java);
- the application implements a DSS Client;
- the DSS Server may be capable of creating a webpage in which the use of the LSCD is handled;

Note that the creation of the webpage by the DSS server is NOT part of the profile; it is left to the implementors of the service.

2.3. Relationship to Other Profiles

The profile in this document is based on the [DSSCore]. The profile in this document may be implemented.

This profile provides means for the explicit management of local signature computations with [DSSCore] and other existing profiles, and as such, it may be used in conjunction with these specifications.

3. Profile of Signing Protocol

3.1. Element <dss:SignRequest>

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

The `<SignRequest>` message SHOULD be authenticated and the integrity be protected by the protocol binding.

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

3.2. Element <dss:SignResponse>

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

4. Protocol Bindings

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

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 OASIS DSS protocol messages defined in this section leave room for such protocol message level security.

OASIS DSS protocol messages can be generated and exchanged using a variety of protocols. The binding specifications under Section 4, "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>
```

4.1. Transport Binding

A HTTP POST binding is defined as a mechanism by which OASIS 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`

4.1.1. Overview

The HTTP POST binding is intended for cases in which the OASIS 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.

4.1.2. 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 OASIS 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 OASIS DSS signing request, then the form control MUST be named `SignRequest`. If the message is a OASIS 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.

The `action` attribute of the form MUST be the recipient's HTTP endpoint for the protocol or profile using this binding to which the OASIS 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.

4.1.3. HTTP and Caching Considerations

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

4.2. Security Binding

4.3. Secure Conversation Security Binding

4.4. Local Signature Computation Profile

When using the `<SignRequest>` and `<SignResponse>` messages defined in Section 3, “Profile of Signing Protocol” 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 4, “Protocol Bindings” SHALL be used. The Section 4.1, “Transport Binding” SHOULD be used in combination with Section 4.2, “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 OASIS 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 OASIS 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

4.4.1. 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 OASIS 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 OASIS 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>
```

4.5. 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 OASIS DSS. This profile is based on the OASIS DSS Asynchronous Processing Abstract Profile [DSSAsync].

For the establishment of a shared secret the Section 4.3, “Secure Conversation Security Binding” can be used. When using this security binding, the lifecycle of the document that is stored in the OASIS DSS temporary document repository SHOULD correspond with the lifecycle of the secure conversation context.

4.5.1. 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 OASIS DSS protocol messages defined in Section 3.1, “Element <dss:SignRequest>” and involves the user agent.

4.5.1.1. SignRequest

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

4.5.1.2. 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 OASIS DSS server.

4.5.2. Signing

This builds on Section 4.4, "Local Signature Computation Profile".

The `<SignRequest>` and `<SignResponse>` messages as defined in Section 3, "Profile of Signing Protocol" SHALL include the document reference. This document reference is used to correlate the document transmitted via Section 4.5.1, "Prepare for signing" and the current user agent session.

4.5.2.1. 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.

4.5.2.2. 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].

4.5.3. Finalize signing

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

4.5.3.1. 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.

4.5.3.2. SignResponse

The response message contains the signed document.

4.6. Original Document Verification Profile

When the signing messages between the OASIS 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 OASIS DSS Verifying Protocol as defined in section 4 of [DSSCore].

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

4.6.1. 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>
```

4.6.2. 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.

4.7. 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 3, “Profile of Signing Protocol” is signed, the `Destination` XML attribute in the root OASIS 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 OASIS DSS protocol defined in this document is similar to the SAML protocol most of the security considerations defined in [SAMLCore] also apply to the OASIS DSS protocol.

As opposed to signatures created in the context of entity authentication, creation of document signatures using the OASIS 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 OASIS 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 OASIS DSS protocol request message that is transferred from relying party to the OASIS DSS server via the intermediate user agent. This in order to avoid undetectable document manipulations by for example the end user. The Section 4.3, “Secure Conversation Security Binding” can ensure this.

5. Conformance

The present profile defines two conformance levels. These two levels are defined in the clauses below.

5.1. Conformance Level 1

Text

5.2. Conformance Level 2

Text

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>
```

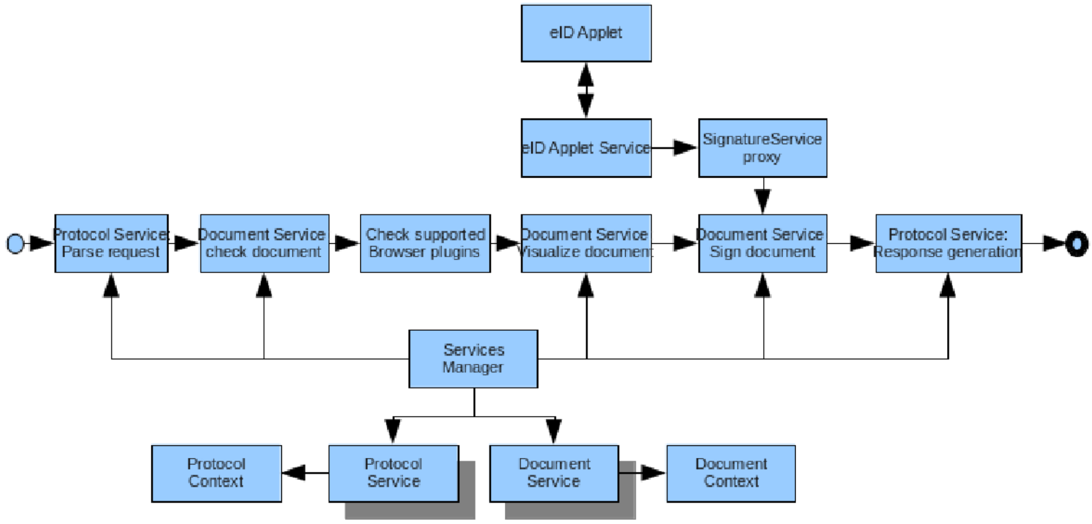
```xml
 <element name="OriginalDocument" type="tns:OriginalDocumentType" />

 <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

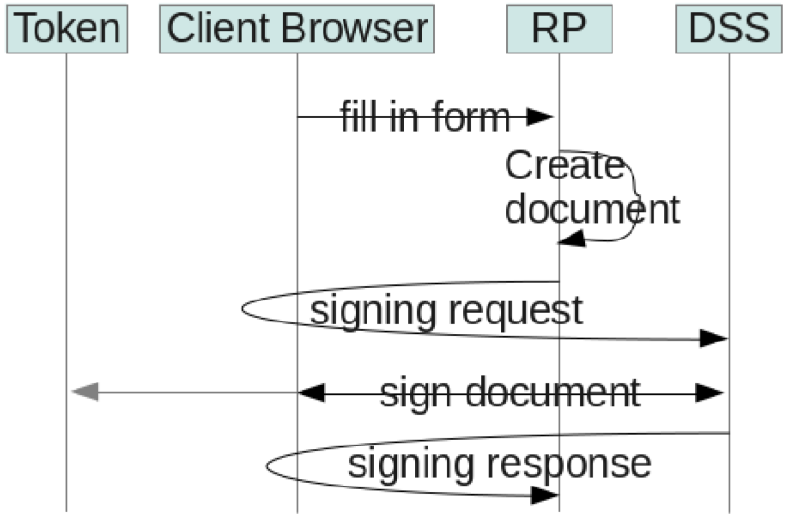


An end-user enters the DSS signature pipeline via some protocol. First of all the appropriate protocol service parses the request. At this step the mime type of the incoming document is determined. Via the mime type the appropriate document service can be selected. The document service will first check 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.		
Revision 0.2.6	April18, 2012	E.J. van Nigtevecht
Added a chapter Background, a chapter Profile Features and a 'reminder' for the chapter Conformance. Reshuffled some chapters into chapter Profile of Signing Protocol and chapter Protocol Bindings. Some textual Improvements.		
Revision 0.3	November 8, 2012	E.J. van Nigtevecht
Re-written for the use of the HTTP-POST mechanism and the use of the asynchronous profile.		