LSS Technical Specification

Table of contents

1 Introduction 3

2 Rendering of signature flows 4

3 Security guidelines 5
3.1 LSS back-end only accessible via SSL 5
3.2 Content Security Policy 5
3.2.1 Inline scripts 5
3.3 Securing JavaScript – Module Pattern 6

4 XML-DSig structure 7
4.1 General requirements 8
4.2 Algorithms 8
4.3 Signature structure 9
4.4 SignedInfo structure 9
4.5 Reference structure 9
4.6 KeyInfo structure 9
4.7 Object structure 10
4.8 SignatureProperty structure 10
4.9 Signature properties 11
4.9.1 The action property 12
4.9.2 The RequestIssuer property 12
4.9.3 The TimeStamp property 12
4.9.4 The signtext property 12
4.9.5 The stylesheetDigest property 12
4.9.6 The stylesheetIdentifier property 13

5 References 14

Version history

December 16th 2016	Version 2.0	TSS
4th April 2014	Version 1.1	TN
28th March 2014	Version 1.0	MSP
27th March 2014	Version 0.97	TN
23rd March 2014	Version 0.96	TSS
20th March 2014	Version 0.95	TN
13th March 2014	Version 0.94	BS
4th March 2014	Version 0.93	TSS
31th January 2014	Version 0.92	MSP
10th January 2014	Version 0.91	MSP
20th December 2013	Version 0.6	MSP
16th December 2013	Version 0.5	MSP

1 Introduction

This document is a technical supplement to the document **"General Technical Specification"** and is intended for LSS suppliers and their technical staff implementing a LSS for NemID backend.

As a supplier of LSS products, it is possible to integrate to the LSS for NemID. This makes it possible for employees in companies with the LSS, to use NemID for business from JavaScript enabled devices such as tablets, smartphones and ordinary computers. Please note, that service providers may or may not choose to support the LSS for NemID functionality in their services.

2 Rendering of signature flows

It is the responsibility of the LSS back-end to validate that the sign texts received for signing flows conform to the specifications provided in DanID's service provider documentation for the specific signing flows (i.e. plain text, HTML, XML and PDF).

A white list of html- and pdf tags allowed can be found in Nets DanID's general service provider documentation. It is required, that the LSS backend perform the same restrictions to signature flows.

Rendering of the document to-be-signed is the responsibility of the LSS backend. It is important that the user is presented with the document to-be-signed correctly. In general, it is the responsibility of the LSS backend to enforce the "What You See Is What You Sign" and by this make sure that the presented document conforms to the signed document returned to the service provider.

3 Security guidelines

As for service providers, LSS suppliers are recommended to follow the security guidelines found in the general service provider documentation from Nets DanID.

The LSS supplier is required to be up-to-date of the security guidelines both from Nets DanID and from the LSS for NemID solution.

3.1 LSS back-end only accessible via SSL

It must be enforced that all connections to the LSS back-end are over https/SSL.

3.2 Content Security Policy

The most notable recommendation is the content security policy http headers.

It is required that the LSS supplier understands and is up-to-date on the current recommendations regarding these headers and take this into account for their LSS back-end implementation.

http://www.html5rocks.com/en/tutorials/security/content-security-policy/
http://www.w3.org/TR/CSP/

3.2.1 Inline scripts

It can be expected that some service provider's use the content security headers to disable inline scripts completely from their web pages. In this scenario only scripts loaded from files from a domain whitelisted by the service provider are permitted. This implies that all JavaScript (and other scripts) must be included by file reference and hosted under the LSS Client URL to ensure that service providers are able to white-list the scripts.

```
//NOT ALLOWED
<script> alert("This is blocked"); </script>

//ALLOWED
<script src="alert.js" />
```

3.3 Securing JavaScript – Module Pattern

It is recommended that the JavaScript deployed at the LSS back-end is executed inside a closure as an anonymous function to prevent global access to any variables or methods inside the closure. This is a general pattern referred to as the *Module Pattern*.

```
//NOT in a closure - func1 is accessible from other JS sources
var func1 = function(){alert("hey");};
func1();

//In a closure - not accessible from other JS sources
(function(){alert("hey");})();
```

4 XML-DSig structure

The signed XML document produced by the LSS must comply with a set of requirements. Here we will specify these requirements.

Note, that some of these requirements arise from the overall desire, that the document should validate using OOAPI [TU].

When implementing the LSS signing it is strongly recommended that the produced documents are validated using OOAPI.

The general form of the signed document the LSS should produce is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<openoces:signature version="0.1">
  <ds:Signature Id="signature">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod .. />
      <ds:SignatureMethod ../>
      <ds:Reference URI="#ToBeSigned">
        <ds:DigestMethod .. />
        <ds:DigestValue>8/bU0v8..</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
      lHPbne...
    </ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>MIIFQTCCAy...</ds:X509Certificate>
      </ds:X509Data>
      <ds:X509Data>
        <ds:X509Certificate>MIIFvzCCBK...</ds:X509Certificate>
      </ds:X509Data>
      <ds:X509Data>
        <ds:X509Certificate>MIIGSDCCBD...</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <ds:Object Id="ToBeSigned">
      <ds:SignatureProperties>
        <ds:SignatureProperty Target="signature">
          <openoces:Name>propertyname</openoces:Name>
          <openoces:Value ..>propertyvalue</openoces:Value>
        </ds:SignatureProperty>
        <ds:SignatureProperty
        ..
        </ds:SignatureProperty>
      </ds:SignatureProperties>
    </ds:Object>
  </ds:Signature>
</openoces:signature>
```

In the following the individual sections in the document will be detailed.

For a description of the XML signing mechanism and semantics we refer to the standard [XMLDSIG].

4.1 General requirements

The document MUST utilize UTF-8 encoding.

The document MUST NOT use XML comments.

The document MUST use namespace prefixes to specify namespaces.

The prefixes used MUST be:

Namespace prefix	**Namespace URI**
ds	http://www.w3.org/2000/09/xmldsig#
openoces	http://www.openoces.org/2006/07/signature#

The *ds:Signature* element MUST conform to the XML-DSig standard [XMLDSIG].

The document MUST have a root element, *openoces:signature* with exactly one child element of type *ds:Signature.*

The root element MUST have an attribute *version* with value *0.1.*

4.2 Algorithms

This section describes the algorithms allowed in OpenSign signatures. The signatures MUST only use algorithms in this table.

Purpose	**Algorithm**
Canonicalization	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature	http://www.w3.org/2001/04/xmldsig-more#rsa-sha256

Digest	http://www.w3.org/2001/04/xmlenc#sha256

4.3 Signature structure

The *ds:Signature* element MUST have an attribute, *Id,* with value *signature.* This element is referred from the *SignatureProperty* elements - see below.

4.4 SignedInfo structure

The *ds:SignedInfo* element specifies exactly which XML-nodes are under signature. This element must therefore have a very specific form, which is specified is this section.

The *SignedInfo* element MUST contain exactly one *Reference* element must be included.

4.5 Reference structure

This *Reference* element MUST NOT define any transforms, i.e. it MUST have exactly two children, *DigestMethod* and *DigestValue*.

The reference must be an id-type referring an element with Id *ToBeSigned.* This means that the URI attribute on the *Reference* element must have the value *#ToBeSigned.* No other values are allowed.

The referred element – having id *ToBeSigned* – MUST be of type *ds:Object*, and must be a child of the *ds:Signature* element.

4.6 KeyInfo structure

The *ds:KeyInfo* element contains the signing certificate chain: The **signing certificate** holds the public key which will decrypt the *SignatureValue.*

KeyInfo contains a sequence of certificates. The sequence MUST be the ordered certificate chain consisting of:[1]

1. Root CA certificate

[1] OOAPI will validate a signature with only the signing certificate supplied. However, the NemID applets have always supplied the full certificate chain, therefore service providers (TU's) may exist that depend on that behavior.

2. Intermediate CA certificate
3. Signing certificate

The certificates MUST be ordered as listed above.

4.7 Object structure

The *ds:Object* element is the XML node-set that is actually signed.

The element MUST define both the *ds* and the *opensign* namespaces.

The element MUST have an *Id* attribute with value *ToBeSigned.*

The element MUST have exactly one *ds:SignatureProperties* child element.

The *ds:SignatureProperties* element MUST consist of a sequence of *ds:SignatureProperty* elements.

4.8 SignatureProperty structure

Each *SignatureProperty* element MUST have an attribute *Target* with value *signature.*

Each *SignatureProperty* element MUST have exactly two children of types *opensign:Name* and *opensign:Value.*

The *opensign:Name* element defines the name of the property in its node-text, the *opensign:Value* element defines the property value.

Each *opensign:Value* element MUST have exactly two attributes:

Value attribute	**Legal value**
Encoding	base64 | xml
VisibleToSigner	yes | no

When *Encoding="base64"* is specified, the node-text MUST be Base64 encoded bytes. If the value should be interpreted as a string (see below), the encoded bytes MUST be a UTF-8 encoding of the string value.

When *Encoding="xml"* is specified, the node-text directly specifies the value of the property.

The *VisibleToSigner* attribute has value *yes* when the property at hand was shown to the user during the signing process – and value *no* otherwise.

How these attribute values should be selected depends on the property specified in *opensign:Name.* These values are specified in detail below.

4.9 Signature properties

Most signature properties are directly related to the parameters passed in the "Parameters" call – see the General Technical Specification document, Section 5. In the table below this relationship is covered by the Pass-through column.

The properties passed in parameter **SIGN_PROPERTIES** MUST be added as *SignatureProperty* elements with value-attributes *Encoding="base64"* and *VisibleToSigner="no".*

The values given in column Flows are a shorthand notation for whether this attribute is used in the logon (L), signing (S) or XML-signing (X) flows, respectively. Here *S* denote the text-, PDF-, and HTML-signing flows.

For example, a property with value *SX* MUST NOT be specified in a logon flow and MUST be specified in any signing-flow.

Columns Encoding and VTS (VisibleToSigner) give the values of the corresponding *opensign:Value* attributes.

Property name	**Flows**	**Pass-through from**	**Encoding**	**VTS**
action	LSX	-	xml	no
RequestIssuer	LSX	REQUESTISSUER	base64	yes
TimeStamp	LSX	TIMESTAMP	base64	no
signtext	SX	SIGNTEXT	base64	yes | no

stylesheetDigest	X	SIGNTEXT_TRANSFORMATION[2]	base64	no
stylesheetIdentifier	X	SIGNTEXT_TRANSFORMATION_ID	xml	no

In the following the semantics of the properties are described.

4.9.1 The *action* property

This property MUST be

- *logon* in a logon flow
- *sign* in a signing flow

4.9.2 The *RequestIssuer* property

The value of this property MUST be taken from the REQUESTISSUER parameter.

4.9.3 The *TimeStamp* property

This value of this property MUST be taken **as is** from the TIMESTAMP parameter.

4.9.4 The *signtext* property

The value of the *signtext* property MUST be taken from the SIGNTEXT parameter.

Note, that in a PDF-signing flow the value is a Base64 encoding of the PDF-bytes.

When the SIGNTEXT_FORMAT is *XML* the *VisibleToSigner* attribute MUST ben *no,* otherwise *yes.*

4.9.5 The *stylesheetDigest* property

In an XML-signing flow, the LSS MUST compute the value of this property as a SHA256 digest of the UTF-8 encoded XSLT style sheet passed in parameter SIGNTEXT_TRANSFORMATION.

[2]The value of stylesheetDigest is a SHA256 digest of the SIGNTEXT_TRANSFORMATION – so not direct passthrough.

4.9.6 The *stylesheetIdentifier* property

The value of this property MUST be taken directly from parameter SIGNTEXT_TRANSFORMATION_ID. If that parameter is not passed, this property MUST be omitted.

5 References

[TU]	DanID TU-pakke https://www.nets-danid.dk/tu-pakke
[XMLDSIG]	XML Signature Syntax and Processing (Second Edition) http://www.w3.org/TR/xmldsig-core/