Digital Signature Service Protocol Specifications

Version 1.3.2

Frank Cornelis

April 26, 2016

Abstract

This document details on the Digital Signature Service Protocol specifications.

1. Introduction

This Digital Signature Service Protocol features inherent security. This means that developers, when integrating the protocol, are forces to operate in a secure way by design. Thus the design of this protocol features certain security properties that integrating parties can no longer circumvent.

In this document we do not give a formal specification of the Digital Signature Service Protocol. Instead we define the protocol by means of a protocol run example. We find it more valuable to explain the choices we made when designing this protocol.

The protocol involves three parties:

- The web application that wants the end-user to sign a certain document.
- The DSS service that facilitates in letting the end-user to sign the document.
- The end-user (web browser client) that actually performs the document signing using for example an eID card.

The signing protocols of the OASIS DSS specification [DSSCore] mainly focus on centralized key management systems. Such an architecture makes sense for situations where the connecting clients do not 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 the creation and validation of signatures within a certain business context might be easier using a centralized platform.
- When transforming a paper-world business work flow to a digital equivalent that includes the creation and/or validation of electronic 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 instances 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 (DSS) systems require a new set of protocol messages for the creation of signatures where signature computation is accomplished via local tokens.

1.1. Namespaces

The XML namespaces used in the following sections are described in Table 1, “XML Namespaces” .

Table 1. XML Namespaces

Prefix	Namespace
dss	urn:oasis:names:tc:dss:1.0:core:schema
async	urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:1.0
ds	http://www.w3.org/2000/09/xmldsig#
md	urn:oasis:names:tc:SAML:2.0:metadata
saml2	urn:oasis:names:tc:SAML:2.0:assertion
soap	http://www.w3.org/2003/05/soap-envelope
wsa	http://www.w3.org/2005/08/addressing
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512
ec	http://www.w3.org/2001/10/xml-exc-c14n#
dssp	urn:be:e-contract:dssp:1.0
vr	urn:oasis:names:tc:dss-x:1.0:profiles:verificationreport:schema#
vs	urn:oasis:names:tc:dssx:1.0:profiles:VisibleSignatures:schema#
wsc	http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512
xsi	http://www.w3.org/2001/XMLSchema-instance
xacmlp	urn:oasis:names:tc:xacml:2.0:policy:schema:os

1.2. References

[Base64] S. Josefsson, The Base16, Base32, and Base64 Data Encodings , The Internet Society, 2006 *http://tools.ietf.org/html/rfc4648 [http://tools.ietf.org/html/rfc4648]*

[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. Hhnlein et al., Profile for Comprehensive Multi-Signature Verification Reports Version 1.0 , OASIS, November 2010 *http://docs.oasis-open.org/dss-x/profiles/verification-report/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]*

[DSSVisSig] Ezer Farhi et al., Visible Signature Profile of the OASIS Digital Signature Services Version 1.0 , OASIS, 8 May 2010 *http://docs.oasis-open.org/dss-x/profiles/visual-sig/v1.0/cs01/oasis-dssx-1.0-profiles-visualsig-cs1.pdf [http://docs.oasis-open.org/dss-x/profiles/visualsig/v1.0/cs01/oasis-dssx-1.0-profiles-visualsig-cs1.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 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.

[SAML] Scott Cantor, John Kemp, Rob Philpott, Eve Maler, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 , OASIS, 15 March 2005 *https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf*

[SAML-MD] Scott Cantor, Jahan Moreh, Rob Philpott, Eve Maler, Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0 , OASIS, 15 March 2005 *https://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf*

[SOAP] W3C, SOAP Version 1.2 , *SOAP Version 1.2 [http://www.w3.org/TR/soap12-part1/]* W3C Recommendation 27 April 2007

[SwA] WS-I, Attachments Profile Version 1.0 , *WS-I Attachments Profile Version 1.0 [http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html]* Web Services Interoperability Organization

[RFC 2246] T. Dierks, C. Allen, The TLS Protocol Version 1.0 , *http://www.ietf.org/rfc/rfc2246.txt* IETF (Internet Engineering Task Force) RFC 2246, January 1999.

[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-Sec] Kelvin Lawrence, Chris Kaler, OASIS Web Services Security: SOAP Message Security 1.1 , OASIS, February 2006 *OASIS WS-Security 1.1 [https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf]*

[WSS-SAML] Anthony Nadalin, Chris Kaler, Ronald Monzillo, Phillip Hallam-Baker OASIS Web Services Security: SAML Token Profile 1.1 , OASIS, February 2006 *OASIS*

WS-Security SAML Profile 1.1 [https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf]

[WSS-UT] Anthony Nadalin, Chris Kaler, Ronald Monzillo, Phillip Hallam-Baker OASIS Web Services Security: UsernameToken Profile 1.1 , OASIS, February 2006 *OASIS WS-Security UsernameToken Profile 1.1 [https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf]*

[WSS-X509] Anthony Nadalin, Chris Kaler, Ronald Monzillo, Phillip Hallam-Baker OASIS Web Services Security: X.509 Certificate Token Profile 1.1 , OASIS, 1 February 2006 *OASIS WS-Security X.509 Certificate Token Profile 1.1 [https://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-x509TokenProfile.pdf]*

[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]*

[XACML] eXtensible Access Control Markup Language (XACML) Version 2.0 , OASIS, 1 Feb 2005 *http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf [http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf]*

[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]*

2. The basic protocol run

The basic protocol run consists of three request/response messages.

- The web application uploads the document to be signed to the DSS.
- Actual signing request from the web application towards the DSS. This involves a web browser POST request/response flow.
- The web application downloads the signed document from the DSS.

2.1. Document uploading

A protocol run starts with a SOAP version 1.2 request [SOAP] from the web application towards the DSS. This first step allows the web application to send over the document to be signed to the

DSS. It also allows for the web application and the DSS to establish a security context. Via this mechanism, subsequent request/response messages are protected.

The document is not transmitted via a Browser POST. Given the upload limitation of most end-user's internet connection, this would otherwise result in a bad end-user experience when trying to sign a large document.

An example initial request message is given below.

```
<soap:Envelope>
    <soap:Body>
        <dss:SignRequest Profile="urn:be:e-contract:dssp:1.0">
            <dss:OptionalInputs>
                <dss:AdditionalProfile>
            urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing
                </dss:AdditionalProfile>
                <wst:RequestSecurityToken>
                    <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>
                <dss:SignaturePlacement WhichDocument="doc1"
                    CreateEnvelopedSignature="true"/>
                <dss:SignatureType>
                    urn:be:e_contract:dssp:signature:xades-x-l
                </dss:SignatureType>
            </dss:OptionalInputs>
            <dss:InputDocuments>
                <dss:Document ID="doc1">
                    <dss:Base64Data MimeType="...">the document</dss:Base64Data>
                </dss:Document>
            </dss:InputDocuments>
        </dss:SignRequest>
    </soap:Body>
</soap:Envelope>
```

The document to be signed is passed via the `<dss:Document>` element. DSS implementations should support both `<dss:Base64XML>` and `<dss:Base64Data>` to transport documents. We use the OASIS Asynchronous Abstract Profile [DSSAsync] given the nature of this first request.

Because the transmitted document can contain sensitive data, the SOAP request is transmitted over SSL [RFC 2246] . This also allows the web application to authenticate and trust the DSS endpoint.

Via WS-SecureConversation [WS-SecConv] the web application and DSS establish a shared secure conversation and corresponding session key. This session key is used to sign subsequent message exchanges. This mechanism thus gives us integrity protection.

We do not use the WS-Addressing features as defined within the WS-SecureConversation specs, as we do not need such routing capabilities.

Since implementing DSS products might support multiple signature types, we include an optional `<dss:SignatureType>` element. We define the following signature type URIs:

- `urn:be:e-contract:dssp:signature:xades-x-l`

  This signature type is compatible with the signatures created by eID DSS 1.0.x.

- `urn:be:e-contract:dssp:signature:xades-baseline`

  ETSI XAdES Baseline profile signatures.

- `urn:be:e-contract:dssp:signature:pades-baseline`

  ETSI PAdES Baseline profile signatures.

If the `<dss:SignatureType>` element is not provided, the DSS will determine the most appropriate signature type itself.

The DSS server responds as follows:

```
<soap:Envelope>
    <soap:Body>
        <dss:SignResponse Profile="urn:be:e-contract:dssp:1.0">
            <dss:Result>
                <dss:ResultMajor>
urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending
                </dss:ResultMajor>
            </dss:Result>
            <dss:OptionalOutputs>
                <async:ResponseID>responseId</async:ResponseID>
                <wst:RequestSecurityTokenResponseCollection>
                    <wst:RequestSecurityTokenResponse>
                        <wst:TokenType>
        http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct
                        </wst:TokenType>
                        <wst:RequestedSecurityToken>
                            <wsc:SecurityContextToken wsu:Id="token-ref">
                                <wsc:Identifier>
                                    token-id
                                </wsc:Identifier>
                            </wsc:SecurityContextToken>
                        </wst:RequestedSecurityToken>
                        <wst:RequestedAttachedReference>
                            <wsse:SecurityTokenReference>
                                <wsse:Reference URI="#token-ref"/>
                            </wsse:SecurityTokenReference>
                        </wst:RequestedAttachedReference>
                        <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>...</wsu:Created>
                            <wsu:Expires>...</wsu:Expires>
                        </wst:Lifetime>
                    </wst:RequestSecurityTokenResponse>
                </wst:RequestSecurityTokenResponseCollection>
            </dss:OptionalOutputs>
        </dss:SignResponse>
    </soap:Body>
</soap:Envelope>
```

The value of the `<async:ResponseID>` element is used by the web application to further reference the uploaded document that has to be signed.

The secure conversation token and corresponding proof-of-possession key is used to secure the subsequent messages between application and DSS server. The proof-of-possession key is generated using the `P_SHA-1` algorithm [RFC 2246] . By using the `P_SHA-1` algorithm, both client and server have some guarantee that the computed shared secret is only used in the context of the current DSS secure conversation and that it offers sufficient secrecy.

2.1.1. Errors

In case the DSS receives an unsupported document format, the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

```
urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
```

and a `<dss:ResultMinor>` of

```
urn:be:e-contract:dssp:1.0:resultminor:UnsupportedMimeType
```

In case the DSS receives an unsupported `<dss:SignatureType>` value, the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

```
urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
```

and a `<dss:ResultMinor>` of

`urn:be:e-contract:dssp:1.0:resultminor:UnsupportedSignatureType`

In case the DSS receives a `<dss:SignatureType>` value that is not supported for the given mime type, the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

`urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError`

and a `<dss:ResultMinor>` of

`urn:be:e-contract:dssp:1.0:resultminor:IncorrectSignatureType`

In case the DSS requires application credentials during the document uploading (e.g. production only usage), the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

`urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError`

and a `<dss:ResultMinor>` of

`urn:be:e-contract:dssp:1.0:resultminor:authentication-required`

2.2. Browser POST

The next request/response messages use a Browser POST as the DSS requires control over the end-user web browser to interact with the end-user and signature creation device (e.g., a smart card). This flow is similar to a classical SAML 2.0 Browser POST sequence.

The web application initiates a Browser POST towards the DSS server by means of the following HTML page [HTML401] :

```
<html>
    <head><title>DSS Browser POST</title></head>
    <body>
        <p>Redirecting to the DSS Server...</p>
        <form name="BrowserPostForm" method="post"
            action="https://www.e-contract.be/dss-ws/start">
            <input type="hidden" name="PendingRequest" value="..."/>
        </form>
        <script type="text/javascript">
            window.onload = function() {
                document.forms["BrowserPostForm"].submit();
            };
        </script>
    </body>
</html>
```

Here the `PendingRequest` field of the HTML form contains the base64 encoded [Base64] `<async:PendingRequest>` message. This `<async:PendingRequest>` message looks as follows:

```
<async:PendingRequest Profile="urn:be:e-contract:dssp:1.0">
    <dss:OptionalInputs>
        <dss:AdditionalProfile>
            urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing
        </dss:AdditionalProfile>
        <async:ResponseID>responseId</async:ResponseID>
        <wsa:MessageID>
            uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
        </wsa:MessageID>
        <wsu:Timestamp>
            <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
            <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
        </wsu:Timestamp>
        <wsa:ReplyTo>
            <wsa:Address>web application landing page URL</wsa:Address>
        </wsa:ReplyTo>
        <ds:Signature>
            <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="">
                    <ds:Transforms>
                        <ds:Transform Algorithm=
                        "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                         <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>
    </dss:OptionalInputs>
</async:PendingRequest>
```

Via the `<async:ResponseID>` element the web application references the document that was previously transmitted to the DSS server via the SOAP call described under Section 2.1, “Document uploading” . The different WS-Addressing and XML signature elements are used to secure the transmitted message. The message-level XML signature is using the security token's corresponding proof-of-possession key. The XML signature requires a `<ds:Reference>` element with attribute `URI=""` to sign the entire `<async:PendingRequest>` document. We use `URI=""` here, as the top-level `<async:PendingRequest>` element does not allow for an identifier attribute (e.g., an `<wsu:Id>` attribute). DSS implementations should verify the XML signature and check whether it corresponds with a previously established secure conversation token during the Section 2.1, “Document uploading” phase of the protocol run.

The actual signing process between DSS server and signature creation device will most likely be using a proprietary protocol. This part is out of scope of this specification document for several reasons:

- This protocol is smart card token specific.
- This protocol might be web browser and/or client platform dependent. This might involve Java Applets, web browser extensions (Google Chrome) and such. Given the volatile web browser landscape, this protocol cannot be standardized (yet).

After signing the document, the DSS server responds with the following HTML page:

```
<html>
    <head><title>DSS Browser POST</title></head>
    <body>
        <p>Redirecting to the web application...</p>
        <form name="BrowserPostForm" method="post"
            action="value of wsa:ReplyTo/wsa:Address element">
            <input type="hidden" name="SignResponse" value="..."/>
        </form>
        <script type="text/javascript">
            window.onload = function() {
                document.forms["BrowserPostForm"].submit();
            };
        </script>
    </body>
</html>
```

Note here that the web application should be ready to accept HTTP POST messages at the address indicated by the `wsa:ReplyTo/wsa:Address` element value.

The `SignResponse` field of the HTML `<form>` element contains the base64 encoded `<dss:SignResponse>` message. This `<dss:SignResponse>` message looks as follows:

```
<dss:SignResponse Profile="urn:be:e-contract:dssp:1.0">
    <dss:Result>
        <dss:ResultMajor>
urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing:resultmajor:Pending
        </dss:ResultMajor>
    </dss:Result>
    <dss:OptionalOutputs>
        <async:ResponseID>responseId</async:ResponseID>
        <wsa:RelatesTo>previous MessageID value</wsa:RelatesTo>
        <wsu:Timestamp>
            <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
            <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
        </wsu:Timestamp>
        <wsa:To>web application landing page URL</wsa:To>
        <ds:Signature>
            <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="">
                    <ds:Transforms>
                        <ds:Transform Algorithm=
            "http://www.w3.org/2000/09/xmldsig#enveloped-signature">
                        <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>
    </dss:OptionalOutputs>
</dss:SignResponse>
```

The DSS server signs the message with the secure conversation token's proof-of-possession key. The web application should check this signature. However, compared to the legacy eID DSS 1.0.x protocol, the consequences of a web application not checking this signature are less exploitable as no vital information is passed as part of this response message.

2.2.1. Errors

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

```
urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
```

and a `<dss:ResultMinor>` of

```
urn:be:e-contract:dssp:1.0: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

```
urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
```

and a `<dss:ResultMinor>` of

```
urn:be:e-contract:dssp:1.0:resultminor:client-runtime
```

2.3. Downloading the signed document

Finally the web application can request the signed document from the DSS server via a SOAP call:

```
<soap:Envelope>
    <soap:Header>
        <wsse:Security soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="timestamp">
                <wsu:Created>...</wsu:Created>
                <wsu:Expires>...</wsu:Expires>
            </wsu:Timestamp>
            <wsc:SecurityContextToken wsu:Id="token-ref">
                    <wsc:Identifier>
                        token-id
                    </wsc:Identifier>
            </wsc:SecurityContextToken>
            <ds:Signature>
                <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="#timestamp">
                        <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:Reference URI="#body">
                        <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 URI="#token-ref" />
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="body">
        <async:PendingRequest Profile="urn:be:e-contract:dssp:1.0">
            <dss:OptionalInputs>
                <dss:AdditionalProfile>
            urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing
                </dss:AdditionalProfile>
                <async:ResponseID>responseId</async:ResponseID>
                <wst:RequestSecurityToken>
                    <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>
                </wst:RequestSecurityToken>
            </dss:OptionalInputs>
        </async:PendingRequest>
    </soap:Body>
</soap:Envelope>
```

The web application has to sign the SOAP request using WS-Security. This gives the DSS the assurance that no other party downloads the signed document except the original web application.

The WS-Security signature should at least cover the `<wsu:Timestamp>` element. Signing the `<soap:Body>` element is optional to allow for interoperability with Windows Communication Foundation. To ensure interoperability with Windows Communication Foundation, DSS implementations should not require compliance with WS-I Basic Security Profile Version 1.1.

The web application should also instantly cancel the security token by piggybacking a `<wst:RequestSecurityToken>` element.

The DSS server returns the signed document within the SOAP response.

```
<soap:Envelope>
    <soap:Body>
        <dss:SignResponse Profile="urn:be:e-contract:dssp:1.0">
            <dss:Result>
                <dss:ResultMajor>
                    urn:oasis:names:tc:dss:1.0:resultmajor:Success
                </dss:ResultMajor>
                <dss:ResultMinor>
    urn:oasis:names:tc:dss:1.0:resultminor:valid:signature:OnAllDocuments
                </dss:ResultMinor>
            </dss:Result>
            <dss:OptionalOutputs>
                <dss:DocumentWithSignature>
                    <dss:Document ID="doc1">
                        <dss:Base64Data MimeType="...">
                            the signed document
                        </dss:Base64Data>
                    </dss:Document>
                </dss:DocumentWithSignature>
                <wst:RequestSecurityTokenResponseCollection>
                    <wst:RequestSecurityTokenResponse>
                        <wst:RequestedTokenCancelled/>
                    </wst:RequestSecurityTokenResponse>
                </wst:RequestSecurityTokenResponseCollection>
            </dss:OptionalOutputs>
            <dss:SignatureObject>
                <dss:SignaturePtr WhichDocument="doc1"/>
            </dss:SignatureObject>
        </dss:SignResponse>
    </soap:Body>
```

```
</soap:Envelope>
```

3. Extensions

We define several extensions on the basic protocol run.

3.1. Signer Identity

The web application cannot always determine in advance which digital identity will be used by the end-user to sign the document. Different strategies are possible to eventually determine this digital identity. The web application can for example verify the signed document via a DSS verification request as defined under section 4 of [DSSCore] for this. Of course, if multiple signature are present on the document, this can cause problems. Via the elements defined within this section we give the web application additional means to determine the signatory's digital identity.

We allow usage of the `<dss:ReturnSignerIdentity>` optional input element and corresponding `<dss:SignerIdentity>` optional output element as defined under section 4.5.7 of [DSSCore] within the context of the Section 2.2, “Browser POST” . Note that these elements were originally defined within the context of signature verification.

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

This extension can be used in combination with the Section 3.6, “Authorization” extension. In case of an authorization error, the `<dss:SignerIdentity>` element contains the user identity that tried to sign but was not authorized.

3.2. Localization

The web application could include the optional input `<dss:Language>` element as defined in [DSS-Core] section 2.8.3 to indicate the preferred language settings to be used by the DSS server as part of the Section 2.2, “Browser POST” request.

3.3. SOAP with Attachments

For large documents, the web application and DSS service can use SOAP with attachments [SwA] . For larger documents, this can have a very positive impact on the performance of the DSS service. In this case the `<dss:Document>` element looks as follows:

```
<dss:Document ID="doc1">
    <dss:AttachmentReference MimeType="..." AttRefURI="cid:...">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>...</ds:DigestValue>
    </dss:AttachmentReference>
</dss:Document>
```

The digest value should always be present given the fact that we sign the SOAP message body.

The DSS server should only use SOAP attachments for the document downloading (see Section 2.3, “Downloading the signed document”) when the web application itself used SOAP attachments for the document uploading (see Section 2.1, “Document uploading”).

3.4. Role/Location

When signing a digital contract, it is important to be able to determine the role of the signatory and the location. For most contracts this can be determined from the contractual context. If this is not the case, the Digital Signature Service Protocol supports explicit role and location indication as part of the qualified signature.

An optional `<vs:VisibleSignatureConfiguration>` [DSSVisSig] element can be added to the `<dss:OptionalInputs>` element as part of the `<async:PendingRequest>` Browser POST message.

```
<vs:VisibleSignatureConfiguration>
    <vs:VisibleSignaturePolicy>
        DocumentSubmissionPolicy
    </vs:VisibleSignaturePolicy>
    <vs:VisibleSignatureItemsConfiguration>
        <vs:VisibleSignatureItem>
            <vs:ItemName>
                SignatureProductionPlace
            </vs:ItemName>
            <vs:ItemValue xsi:type="vs:ItemValueStringType">
                <vs:ItemValue>
                    Vilvoorde
                </vs:ItemValue>
            </vs:ItemValue>
        </vs:VisibleSignatureItem>
        <vs:VisibleSignatureItem>
            <vs:ItemName>
                SignatureReason
            </vs:ItemName>
            <vs:ItemValue xsi:type="vs:ItemValueStringType">
                <vs:ItemValue>
                    CEO
                </vs:ItemValue>
            </vs:ItemValue>
        </vs:VisibleSignatureItem>
    </vs:VisibleSignatureItemsConfiguration>
</vs:VisibleSignatureConfiguration>
```

3.5. Application Credentials

If required, the web application can authenticate itself by means of a WS-Security [WS-Sec] SOAP header. This allows the DSS to operate in two modes:

- Development mode, where the web applications can access the DSS unauthorized.

• Production mode, where the web applications must be authorized by the DSS.

As part of the document upload, see Section 2.1, “Document uploading” , the application can provide credentials. A Digital Signature Service implementation could use these application credentials for branding or accounting purposes. A Digital Signature Service implementation can implement different types of application credentials. These are detailed in the following sub-sections.

3.5.1. Username/password Application Credentials

Username/password application credentials are provided by means of a WS-Security UsernameToken [WSS-UT] within the SOAP header element. DSS implementations should support both `#PasswordText` and `#PasswordDigest` according to the [WSS-UT] specs.

An example for `#PasswordText` is given below.

```
<wsse:Security soap:mustUnderstand="true">
    <wsse:UsernameToken>
        <wsse:Username>
            username
        </wsse:Username>
        <wsse:Password Type=
                     "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordText">
            password
        </wsse:Password>
    </wsse:UsernameToken>
    <wsu:Timestamp>
        <wsu:Created>
            2014-04-22T13:45:29.956Z
        </wsu:Created>
        <wsu:Expires>
            2014-04-22T13:46:29.956Z
        </wsu:Expires>
    </wsu:Timestamp>
</wsse:Security>
```

An example for `#PasswordDigest` is given below.

```
<wsse:Security soap:mustUnderstand="true">
    <wsse:UsernameToken>
        <wsse:Username>
            username
        </wsse:Username>
        <wsse:Password Type=
                     "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordDigest">
            ...
        </wsse:Password>
        <wsse:Nonce EncodingType=
                       "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary">
            ...
        </wsse:Nonce>
        <wsu:Created>2016-02-29T09:05:24.975Z</wsu:Created>
    </wsse:UsernameToken>
    <wsu:Timestamp>
        <wsu:Created>
            2014-04-22T13:45:29.956Z
        </wsu:Created>
        <wsu:Expires>
            2014-04-22T13:46:29.956Z
        </wsu:Expires>
    </wsu:Timestamp>
</wsse:Security>
```

3.5.2. X509 Application Credentials

X509 certificate application credentials are provided by means of a WS-Security X.509 Certificate Token [WSS-X509] within the SOAP header element. An example is given below.

```
<wsse:Security soap:mustUnderstand="true">
    <wsu:Timestamp wsu:Id="TS">
        <wsu:Created>2016-02-11T20:50:22.711Z</wsu:Created>
        <wsu:Expires>2016-02-11T20:51:22.711Z</wsu:Expires>
    </wsu:Timestamp>
    <wsse:BinarySecurityToken EncodingType=
                         "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary"
        ValueType=
       "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
        wsu:Id="X509">
        ...
    </wsse:BinarySecurityToken>
    <ds:Signature>
        <ds:SignedInfo>
            <ds:CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#soap-body-id">
                <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:Reference URI="#TS">
                <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 URI="#X509"
                      ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
token-profile-1.0#X509v3"/>
            </wsse:SecurityTokenReference>
        </ds:KeyInfo>
    </ds:Signature>
</wsse:Security>
```

The WS-Security signature should at least cover the `<wsu:Timestamp>` element. Signing the `<soap:Body>` element is optional to allow for interoperability with Windows Communication Foundation. To ensure interoperability with Windows Communication Foundation, DSS implementations should not require compliance with WS-I Basic Security Profile Version 1.1.

The corresponding X509 certificate is included by means of a `<wsse:BinarySecurityToken>` element. The XML signature `<ds:KeyInfo>` contains a reference to this binary security token.

3.5.3. SAML Bearer Token Application Credentials

SAML Bearer Token application credentials are provided by means of a WS-Security SAML Token [WSS-SAML] within the SOAP header element.

This type of application credential can be used in a scenario where a back-end system explicitly wants to give a front-end application authorization to initiate a signing operation towards a DSS instance.

An example is given below.

```
<wsse:Security soap:mustUnderstand="true">
    <wsu:Timestamp wsu:Id="TS">
        <wsu:Created>2016-02-11T20:50:22.711Z</wsu:Created>
        <wsu:Expires>2016-02-11T20:51:22.711Z</wsu:Expires>
    </wsu:Timestamp>
    <saml2:Assertion ID="assertion"
        IssueInstant="2016-02-25T12:57:30.359Z" Version="2.0">
        <saml2:Issuer>SAML Issuer</saml2:Issuer>
        <ds:Signature>
            <ds:SignedInfo>
                <ds:CanonicalizationMethod
                    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                <ds:SignatureMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                <ds:Reference URI="#assertion">
                    <ds:Transforms>
                        <ds:Transform Algorithm=
                            "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                        <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>
                <ds:X509Data>
                    <ds:X509Certificate>
                        ...
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </ds:Signature>
        <saml2:Subject>
            <saml2:NameID>Subject Name</saml2:NameID>
            <saml2:SubjectConfirmation Method=
                "urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
        </saml2:Subject>
    </saml2:Assertion>
</wsse:Security>
```

The SAML assertion should at least have the following elements:

- `<saml2:Issuer>`

  Containing a human readable name of the SAML issuer.

- `<ds:Signature:>`

  The SAML assertion should be signed with an X509 certificate trusted by the DSS implementation. The certificate should be included within the `<ds:KeyInfo>` element.

- `<saml2:Subject>`

  Containing a human readable `<saml2:NameID>` subject name.

- `<saml2:SubjectConfirmation>`

  With `Method` value `urn:oasis:names:tc:SAML:2.0:cm:bearer`.

DSS implementations should base their trust on the X509 certificate within the SAML signature `<ds:KeyInfo>`.

3.5.4. SAML Holder-of-key Token Application Credentials

SAML Holder-of-key Token application credentials are provided by means of a WS-Security SAML Token [WSS-SAML] within the SOAP header element.

This type of application credential can be used in a scenario where a back-end system explicitly wants to give a front-end application authorization to initiate a signing operation towards a DSS instance.

Compared with Section 3.5.3, “SAML Bearer Token Application Credentials” the holder-of-key construct gives the back-end system extra assurance that only the front-end application can consume the SAML token towards the DSS instance.

An example is given below.

```
<wsse:Security soap:mustUnderstand="true">
    <wsu:Timestamp wsu:Id="TS">
        <wsu:Created>2016-02-11T20:50:22.711Z</wsu:Created>
        <wsu:Expires>2016-02-11T20:51:22.711Z</wsu:Expires>
    </wsu:Timestamp>
    <saml2:Assertion ID="assertion"
        IssueInstant="2016-02-25T12:57:30.359Z" Version="2.0">
        <saml2:Issuer>SAML Issuer</saml2:Issuer>
        <ds:Signature>
            <ds:SignedInfo>
                <ds:CanonicalizationMethod
                    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                <ds:SignatureMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                <ds:Reference URI="#assertion">
                    <ds:Transforms>
                        <ds:Transform Algorithm=
                            "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                        <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>
                <ds:X509Data>
                    <ds:X509Certificate>
                        ...
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </ds:Signature>
        <saml2:Subject>
            <saml2:NameID>Subject Name</saml2:NameID>
            <saml2:SubjectConfirmation Method=
                "urn:oasis:names:tc:SAML:2.0:cm:holder-of-key">
                <saml2:SubjectConfirmationData>
                    <ds:KeyInfo>
                        <ds:KeyValue>
                            <ds:RSAKeyValue>
                                <ds:Modulus>
                                    ...
                                </ds:Modulus>
                                <ds:Exponent>
                                    ...
                                </ds:Exponent>
                            </ds:RSAKeyValue>
                        </ds:KeyValue>
                    </ds:KeyInfo>
                </saml2:SubjectConfirmationData>
            </saml2:SubjectConfirmation>
        </saml2:Subject>
    </saml2:Assertion>
    <ds:Signature>
        <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm=
                "http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod Algorithm=
                "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#soap-body-id">
                <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:Reference URI="#TS">
                <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 wsse11:TokenType=
    "http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
                <wsse:KeyIdentifier ValueType=
    "http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
                    assertion
                </wsse:KeyIdentifier>
            </wsse:SecurityTokenReference>
        </ds:KeyInfo>
    </ds:Signature>
</wsse:Security>
```

The SAML assertion should at least have the following elements:

- `<saml2:Issuer>`

Containing a human readable name of the SAML issuer.

- `<ds:Signature:>`

  The SAML assertion should be signed with an X509 certificate trusted by the DSS implementation. The certificate should be included within the `<ds:KeyInfo>` element.

- `<saml2:Subject>`

  Containing a human readable `<saml2:NameID>` subject name.

- `<saml2:SubjectConfirmation>`

  With `Method` value `urn:oasis:names:tc:SAML:2.0:cm:holder-of-key`.

- `<saml2:SubjectConfirmationData>`

  Containing the public part of the proof-of-possession key.

DSS implementations should base their trust on the X509 certificate within the SAML signature `<ds:KeyInfo>`.

The WS-Security signature should at least cover the `<wsu:Timestamp>` element. Signing the `<soap:Body>` element is optional to allow for interoperability with Windows Communication Foundation. To ensure interoperability with Windows Communication Foundation, DSS implementations should not require compliance with WS-I Basic Security Profile Version 1.1.

The WS-Security signature should be validated against the proof-of-possession public key within the holder-of-key SAML assertion. The WS-Security signature refers to the SAML assertion via a `<wsse:SecurityTokenReference>` element within its `<ds:KeyInfo>` element.

3.6. Authorization

Applications that want to limit the subjects that are allowed to sign the given document can use the authorization extension. The authorization policy is defined by means of an OASIS XACML 2.0 [XACML] `<xacmlp:Policy>` element within the `<dss:OptionalInputs>` element as part of the `<async:PendingRequest>` Browser POST message.

An example of such a `<xacmlp:Policy>` element is given below.

```
<xacmlp:Policy
    PolicyId="urn:whatever"
    RuleCombiningAlgId=
    "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <xacmlp:Target />
    <xacmlp:Rule RuleId="whatever" Effect="Permit">
        <xacmlp:Target>
            <xacmlp:Subjects>
                <xacmlp:Subject>
                    <xacmlp:SubjectMatch
                        MatchId=
                "urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match">
```

```
                        <xacmlp:AttributeValue DataType=
                        "http://www.w3.org/2001/XMLSchema#string">
                            CN=fcorneli,.*,C=BE
                        </xacmlp:AttributeValue>
                        <xacmlp:SubjectAttributeDesignator
                            AttributeId=
                            "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                            DataType=
                            "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
                    </xacmlp:SubjectMatch>
                </xacmlp:Subject>
                <xacmlp:Subject>
                    <xacmlp:SubjectMatch
                        MatchId=
                        "urn:oasis:names:tc:xacml:1.0:function:x500Name-equal">
                        <xacmlp:AttributeValue DataType=
                        "urn:oasis:names:tc:xacml:1.0:data-type:x500Name">
                            CN=fcorneli2, C=BE
                        </xacmlp:AttributeValue>
                        <xacmlp:SubjectAttributeDesignator
                            AttributeId=
                            "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                            DataType=
                            "urn:oasis:names:tc:xacml:1.0:data-type:x500Name" />
                    </xacmlp:SubjectMatch>
                </xacmlp:Subject>
            </xacmlp:Subjects>
            <xacmlp:Resources>
                <xacmlp:Resource>
                    <xacmlp:ResourceMatch MatchId=
                    "urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                        <xacmlp:AttributeValue DataType=
                        "http://www.w3.org/2001/XMLSchema#anyURI">
                            urn:be:e-contract:dss
                        </xacmlp:AttributeValue>
                        <xacmlp:ResourceAttributeDesignator
                            AttributeId=
                            "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                            DataType=
                            "http://www.w3.org/2001/XMLSchema#anyURI" />
                    </xacmlp:ResourceMatch>
                </xacmlp:Resource>
            </xacmlp:Resources>
            <xacmlp:Actions>
                <xacmlp:Action>
                    <xacmlp:ActionMatch MatchId=
                    "urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <xacmlp:AttributeValue DataType=
                        "http://www.w3.org/2001/XMLSchema#string">
                            sign
                        </xacmlp:AttributeValue>
                        <xacmlp:ActionAttributeDesignator
                            AttributeId=
                            "urn:oasis:names:tc:xacml:1.0:action:action-id"
                            DataType=
                            "http://www.w3.org/2001/XMLSchema#string" />
                    </xacmlp:ActionMatch>
                </xacmlp:Action>
            </xacmlp:Actions>
```

```
        </xacmlp:Target>
    </xacmlp:Rule>
</xacmlp:Policy>
```

DSS implementations that support this extension should recognize at least a resource with identifier `urn:oasis:names:tc:xacml:1.0:resource:resource-id` with a value of `urn:be:e-contract:dss` and an action with identifier `urn:oasis:names:tc:xacml:1.0:action:action-id` with a value of `sign` .

DSS implementations should make sure that they correctly support the `urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match` function as a relying party might not always know the entire distinguished name of the signatory's certificate.

The following subject attributes should be supported by DSS implementations:

- `urn:oasis:names:tc:xacml:1.0:subject:subject-id`

  Indicates the signatory's certificate subject name.

- `urn:be:e-contract:dss:eid:card-number`

  Indicates the signatory's eID card number. This attribute is of type `http://www.w3.org/2001/XMLSchema#string`.

By using XACML we can easily extend the authorization expressions as XACML is a functional complete language. Depending on the business use-case, DSS implementations can define additional attributes to enhance the expressiveness of the authorization restrictions.

An example of a `<xacmlp:Policy>` policy that denied signature creation for a specific subject is given below.

```
<xacmlp:Policy
    PolicyId="urn:whatever"
    RuleCombiningAlgId=
    "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <xacmlp:Target>
        <xacmlp:Resources>
            <xacmlp:Resource>
                <xacmlp:ResourceMatch MatchId=
                "urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
                    <xacmlp:AttributeValue DataType=
                    "http://www.w3.org/2001/XMLSchema#anyURI">
                        urn:be:e-contract:dss
                    </xacmlp:AttributeValue>
                    <xacmlp:ResourceAttributeDesignator AttributeId=
                    "urn:oasis:names:tc:xacml:1.0:resource:resource-id"
                    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
                </xacmlp:ResourceMatch>
            </xacmlp:Resource>
        </xacmlp:Resources>
        <xacmlp:Actions>
            <xacmlp:Action>
```

```
                <xacmlp:ActionMatch MatchId=
                "urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <xacmlp:AttributeValue DataType=
                    "http://www.w3.org/2001/XMLSchema#string">
                        sign
                    </xacmlp:AttributeValue>
                    <xacmlp:ActionAttributeDesignator AttributeId=
                    "urn:oasis:names:tc:xacml:1.0:action:action-id"
                    DataType="http://www.w3.org/2001/XMLSchema#string"/>
                </xacmlp:ActionMatch>
            </xacmlp:Action>
        </xacmlp:Actions>
    </xacmlp:Target>
    <xacmlp:Rule RuleId="allow-all-rule" Effect="Permit"/>
    <xacmlp:Rule RuleId="deny-specific-certificate" Effect="Deny">
        <xacmlp:Target>
            <xacmlp:Subjects>
                <xacmlp:Subject>
                    <xacmlp:SubjectMatch MatchId=
                    "urn:oasis:names:tc:xacml:2.0:function:x500Name-regexp-match">
                        <xacmlp:AttributeValue DataType=
                        "http://www.w3.org/2001/XMLSchema#string">
                            CN=NotAuthorizedSubject
                        </xacmlp:AttributeValue>
                        <xacmlp:SubjectAttributeDesignator AttributeId=
                        "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                        DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name"/>
                    </xacmlp:SubjectMatch>
                </xacmlp:Subject>
            </xacmlp:Subjects>
        </xacmlp:Target>
    </xacmlp:Rule>
</xacmlp:Policy>
```

3.6.1. Errors

In case the end-user was not authorized to sign, the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

`urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError`

and a `<dss:ResultMinor>` of

`urn:be:e-contract:dssp:1.0:resultminor:subject-not-authorized`

3.7. Visible Signatures

Certain signature types, like PAdES signatures, support signature visualization within the document itself. The web application can specify the properties for visualization of the signature. This extension is using [DSSVisSig] and can be part of the Section 3.4, “Role/Location” element. An example is given below.

```
<vs:VisibleSignatureConfiguration>
    <vs:VisibleSignaturePolicy>
```

```
        DocumentSubmissionPolicy
    </vs:VisibleSignaturePolicy>
    <vs:VisibleSignaturePosition
        xsi:type="vs:PixelVisibleSignaturePositionType">
        <vs:PageNumber>1</vs:PageNumber>
        <vs:x>10</vs:x>
        <vs:y>20</vs:y>
    </vs:VisibleSignaturePosition>
    <vs:VisibleSignatureItemsConfiguration>
        <vs:VisibleSignatureItem>
            <vs:ItemName>
                SignerImage
            </vs:ItemName>
            <vs:ItemValue xsi:type="vs:ItemValueURIType">
                <vs:ItemValue>
                    urn:be:e-contract:dssp:1.0:vs:si:eid-photo
                </vs:ItemValue>
            </vs:ItemValue>
        </vs:VisibleSignatureItem>
    </vs:VisibleSignatureItemsConfiguration>
</vs:VisibleSignatureConfiguration>
```

Via the `SignerImage` item the web application can specify different ways to visualize the signature within the document. A DSS implementation should at least support the following profiles:

- An item value `urn:be:e-contract:dssp:1.0:vs:si:eid-photo` indicates that the eID photo should be used as visual element.
- An item value `urn:be:e-contract:dssp:1.0:vs:si:eid-photo:signer-info` indicates that the eID photo should be used as visual element, together with information about the signer (name, role, location, custom code).

DSS implementations can provide custom `SignerImage` URIs towards their connecting customer applications to provide specific signature visualization profiles. Some of these signature visualization profiles can be offered for free, while others might be restricted by means of Section 3.5, “Application Credentials” and are thus only available for specific applications. Signature visualization profiles might also require extra parameters, for example by means of the `CustomText` item. If your application would like to use a company specific signature visualization, please contact your DSS vendor for more information.

3.8. RelayState

Digital Signature Service implementations should implement the `RelayState` parameter. The `RelayState` parameter is used as part of the Section 2.2, “Browser POST” messages.

If the Section 2.2, “Browser POST” request message contains a `RelayState` parameter, the DSS must place the exact `RelayState` data it received with the request into the corresponding `RelayState` parameter in the response.

A DSS implementation should take special care of correct escaping the `RelayState` value within the Section 2.2, “Browser POST” response message.

An integrating web application should not blindly use the value of the received `RelayState` in the context of for example further redirects.

Nonetheless we used to original SAML 2.0 naming for the `RelayState` parameter, this parameter should be used in a similar fashion as the `state` parameter within OAuth 2.

3.9. Application Authorization

This extension can be used within the SAML assertions as defined under Section 3.5.3, “SAML Bearer Token Application Credentials” and Section 3.5.4, “SAML Holder-of-key Token Application Credentials”.

Via this extension back-end systems can restrict front-end applications as to which document(s) they are authorized to get signed.

The back-end system can add the following SAML statement within the SAML assertions to express those restrictions.

```
<saml2:Assertion ID="assertion">
    ...
    <saml2:AuthzDecisionStatement Decision="Permit"
        Resource="urn:be:e-contract:dssp:document:digest:sha-256:...">
        <saml2:Action Namespace="urn:be:e-contract:dssp">
            sign
        </saml2:Action>
    </saml2:AuthzDecisionStatement>
</saml2:Assertion>
```

The `Resource` attribute contains the hexadecimal encoded SHA-256 digest value of the document. The `Resource` attribute value is prefixed with `urn:be:e-contract:dssp:document:digest:sha-256:`. Multiple `<saml2:AuthzDecisionStatement>` elements are allowed.

DSS implementations should verify the digest value of the uploaded document during the Section 2.1, “Document uploading” request against the digest value provided within the SAML assertion `<saml2:AuthzDecisionStatement>` element. In case of a mismatch, the DSS should abort the signing process.

DSS implementations should ignore `<saml2:AuthzDecisionStatement>` elements where the value of the `Namespace` attribute on the `<saml2:Action>` element is different from `urn:be:e-contract:dssp`.

3.9.1. Errors

In case the application is not authorized (i.e., there is a document digest mismatch), the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

```
urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
```

and a `<dss:ResultMinor>` of

`urn:be:e-contract:dssp:1.0:resultminor:subject-not-authorized`

3.10. Attestation

An attestation is issued by DSS implementations upon request by integrating applications. The attestation is basically a statement by the DSS implementation regarding a signature process. The attestation is provided as a SAML 2.0 assertion [SAML] and is digitally signed by the DSS instance.

An integrating web application can request an attestation during the Section 2.1, “Document uploading” request by adding the following element to `<dss:OptionalInputs>`.

```
<dssp:AttestationRequest/>
```

When an attestation has been requested, the DSS implementation will add the following element to `<dss:OptionalOutputs>` as part of the Section 2.3, “Downloading the signed document” response.

```
<dssp:AttestationResponse>
    <saml:Assertion ID="assertion"
        IssueInstant="2016-02-28T08:56:37.789Z" Version="2.0">
        <saml:Issuer>...</saml:Issuer>
        <ds:Signature>
            <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm=
                    "http://www.w3.org/2001/10/xml-exc-c14n#"/>
                <ds:SignatureMethod Algorithm=
                    "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
                <ds:Reference URI="#assertion">
                    <ds:Transforms>
                        <ds:Transform Algorithm=
                            "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                        <ds:Transform Algorithm=
                            "http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    </ds:Transforms>
                    <ds:DigestMethod Algorithm=
                        "http://www.w3.org/2001/04/xmlenc#sha256"/>
                    <ds:DigestValue>
                        ...
                    </ds:DigestValue>
                </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>
                ...
            </ds:SignatureValue>
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>
                        ...
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </ds:Signature>
        <saml:AttributeStatement>
            <saml:Attribute Name=
                "urn:be:e-contract:dssp:attestation:document:digest:sha-256">
                <saml:AttributeValue xsi:type="xs:base64Binary">
                    ...
                </saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name=
                "urn:be:e-contract:dssp:attestation:signed-document:digest:sha-256">
                <saml:AttributeValue xsi:type="xs:base64Binary">
                    ...
                </saml:AttributeValue>
            </saml:Attribute>
        </saml:AttributeStatement>
    </saml:Assertion>
</dssp:AttestationResponse>
```

The attestation SAML assertion is digitally signed by the DSS instance. As attestation SAML assertions will most likely be stored over the long-term, the digest algorithm used within the `<ds:Reference>` elements should be at least `http://www.w3.org/2001/04/xmlenc#sha256`. Similar, the signature method should at least be `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256`. The corresponding RSA key should have a minimum bit size of 2048. The DSS signing certificate should be provided within the `<ds:KeyInfo>` element of the XML signature.

Special care should be taken to make sure that the signature of the SAML assertion remains valid out of the context of the Section 2.3, “Downloading the signed document” SOAP response message. When the receiving web application cuts out the attestation out of the Section 2.3, “Downloading the signed document” response message for long-term storage, the XML signature on the attestation SAML assertion should remain valid.

The following attributes must be provided within a `<saml:AttributeStatement>` element as part of the SAML assertion.

- `urn:be:e-contract:dssp:attestation:document:digest:sha-256`

  Contains the SHA-256 digest value of the original document to be signed. The attribute value is of type `xs:base64Binary`.

- `urn:be:e-contract:dssp:attestation:signed-document:digest:sha-256`

  Contains the SHA-256 digest value of the signed document. The attribute value is of type `xs:base64Binary`.

3.11. Metadata

A DSS implementation should provide the following SAML 2.0 Metadata document [SAML-MD].

```
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor entityID=
    "https://www.e-contract.be/dss-ws/dss-metadata.xml">
    <md:RoleDescriptor protocolSupportEnumeration="urn:be:e-contract:dssp"
        xsi:type="dssp:DigitalSignatureServiceDescriptorType">
        <md:KeyDescriptor use="signing">
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>
                        ...
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </md:KeyDescriptor>
        <dssp:WebServiceEndpoint>
            <wsa:EndpointReference>
                <wsa:Address>
                    https://www.e-contract.be/dss-ws/dss
                </wsa:Address>
            </wsa:EndpointReference>
        </dssp:WebServiceEndpoint>
        <dssp:BrowserPostEndpoint>
            <wsa:EndpointReference>
                <wsa:Address>
                    https://www.e-contract.be/dss-ws/start
                </wsa:Address>
            </wsa:EndpointReference>
        </dssp:BrowserPostEndpoint>
    </md:RoleDescriptor>
</md:EntityDescriptor>
```

The `<md:KeyDescriptor>` should contain the DSS signing key used for Section 3.10, “Attestation” SAML assertion signing.

The address indicated in the `<dssp:WebServiceEndpoint>` element should correspond with the location of the DSS SOAP web service used within Section 2.1, “Document uploading” and Section 2.3, “Downloading the signed document”.

The address indicated in the `<dssp:BrowserPostEndpoint>` element should correspond with the location of the DSS endpoint used within Section 2.2, “Browser POST”.

This metadata document can be used to ease configuration of integrating web applications.

4. Signature Verification

Compared to the eID DSS 1.0.x product, we also made several improvements to the signature verification web service. Here we also allow the usage of SOAP attachments to improve the performance.

The SOAP request for a signature verification looks as follows:

```
<soap:Envelope>
```

```
    <soap:Body>
        <dss:VerifyRequest Profile="urn:be:e-contract:dssp:1.0">
            <dss:OptionalInputs>
                <vr:ReturnVerificationReport>
                    <vr:IncludeVerifier>false</vr:IncludeVerifier>
                    <vr:IncludeCertificateValues>
                        true
                    </vr:IncludeCertificateValues>
                </vr:ReturnVerificationReport>
            </dss:OptionalInputs>
            <dss:InputDocuments>
                <dss:Document ID="document-id">
                    <dss:Base64Data MimeType="text/plain">
                        ...
                    </dss:Base64Data>
                </dss:Document>
            </dss:InputDocuments>
        </dss:VerifyRequest>
    </soap:Body>
</soap:Envelope>
```

We use the OASIS Profile for Comprehensive Multi-Signature Verification Reports Version 1.0 [DSSVer] .

The corresponding SOAP response looks as follows:

```
<soap:Envelope>
    <soap:Body>
        <dss:Response RequestID="the original request id">
            <dss:Result>
                <dss:ResultMajor>
                    urn:oasis:names:tc:dss:1.0:resultmajor:Success
                </dss:ResultMajor>
            </dss:Result>
            <dss:OptionalOutputs>
                <vr:VerificationReport>
                    <vr:IndividualReport>
                        <vr:SignedObjectIdentifier>
                            <vr:SignedProperties>
                                <vr:SignedSignatureProperties>
                                    <xades:SigningTime>
                                        2010-09-13T15:35:49.767+02:00
                                    </xades:SigningTime>
                                    <vr:Location>
                                        Vilvoorde
                                    </vr:Location>
                                    <vr:SignerRole>
                                        <vr:ClaimedRoles>
                                            <xades:ClaimedRole>
                                                CEO
                                            </xades:ClaimedRole>
                                        </vr:ClaimedRoles>
                                    </vr:SignerRole>
                                </vr:SignedSignatureProperties>
                            </vr:SignedProperties>
                        </vr:SignedObjectIdentifier>
                        <dss:Result>
                            <dss:ResultMajor>
                                urn:oasis:names:tc:dss:1.0:resultmajor:Success
                            </dss:ResultMajor>
                        </dss:Result>
                        <vr:Details>
                            <vr:DetailedSignatureReport>
                                <vr:FormatOK>
                                    <vr:ResultMajor>
                                        urn:oasis:names:tc:dss:1.0:detail:valid
                                    </vr:ResultMajor>
                                </vr:FormatOK>
                                <vr:SignatureOK>
                                    <vr:SigMathOK>
                                        <vr:ResultMajor>
                                            urn:oasis:names:tc:dss:1.0:detail:valid
                                        </vr:ResultMajor>
                                    </vr:SigMathOK>
                                </vr:SignatureOK>
                                <vr:CertificatePathValidity>
                                    <vr:PathValiditySummary>
                                        <vr:ResultMajor>
                                            urn:oasis:names:tc:dss:1.0:detail:valid
                                        </vr:ResultMajor>
                                    </vr:PathValiditySummary>
                                    <vr:CertificateIdentifier>
                                        <ds:X509IssuerName>
                                            SERIALNUMBER=201010, CN=Citizen CA, C=BE
                                        </ds:X509IssuerName>
                                        <ds:X509SerialNumber>
                                            21267647932559581687500187050203912999
                                        </ds:X509SerialNumber>
                                    </vr:CertificateIdentifier>
                                    <vr:PathValidityDetail>
                                        <vr:CertificateValidity>
                                            <vr:CertificateIdentifier>
                                                <ds:X509IssuerName>
                                                    SERIALNUMBER=201010, CN=Citizen CA, C=BE
                                                </ds:X509IssuerName>
                                                <ds:X509SerialNumber>
                                                    21267647932559581687500187050203912999
                                                </ds:X509SerialNumber>
                                            </vr:CertificateIdentifier>
                                            <vr:Subject>
                                                cert subject
                                            </vr:Subject>
                                            <vr:ChainingOK>
                                                <vr:ResultMajor>
                                                    urn:oasis:names:tc:dss:1.0:detail:valid
                                                </vr:ResultMajor>
                                            </vr:ChainingOK>
                                            <vr:ValidityPeriodOK>
                                                <vr:ResultMajor>
                                                    urn:oasis:names:tc:dss:1.0:detail:valid
                                                </vr:ResultMajor>
                                            </vr:ValidityPeriodOK>
                                            <vr:ExtensionsOK>
                                                <vr:ResultMajor>
                                                    urn:oasis:names:tc:dss:1.0:detail:valid
                                                </vr:ResultMajor>
                                            </vr:ExtensionsOK>
                                            <vr:CertificateValue>
                                                ...
                                            </vr:CertificateValue>
                                            <vr:SignatureOK>
                                                <vr:SigMathOK>
                                                    <vr:ResultMajor>
                                                        urn:oasis:names:tc:dss:1.0:detail:valid
                                                    </vr:ResultMajor>
                                                </vr:SigMathOK>
                                            </vr:SignatureOK>
                                            <vr:CertificateStatus>
                                                <vr:CertStatusOK>
                                                    <vr:ResultMajor>
                                                        urn:oasis:names:tc:dss:1.0:detail:valid
                                                    </vr:ResultMajor>
                                                </vr:CertStatusOK>
                                            </vr:CertificateStatus>
                                        </vr:CertificateValidity>
                                    </vr:PathValidityDetail>
                                </vr:CertificatePathValidity>
                            </vr:DetailedSignatureReport>
                        </vr:Details>
                    </vr:IndividualReport>
                </vr:VerificationReport>
                <dssp:TimeStampRenewal Before="2013-11-08T08:49:51.040Z"/>
            </dss:OptionalOutputs>
        </dss:Response>
    </soap:Body>
</soap:Envelope>
```

Per signature within the document, you have a `<vr:IndividualReport>` element. The signature is uniquely identified by the `<xades:SigningTime>` element. The certificate of the signatory is delivered via the `<vr:CertificateValue>` element.

The optional `<vr:Location>` element value is extracted from the PAdES `Location` field, or from the XAdES `SignatureProductionPlace/City` element. The optional `<xades:ClaimedRole>` element value is extracted from the PAdES `Reason` field, or from the XAdES `<xades:ClaimedRole>` element.

Via the `<dssp:TimeStampRenewal>` element the web application gets informed by the DSS when the document signatures should be upgraded for long-term validity. See also Section 5, “Long-term validity of signatures” .

4.1. Errors

In case the DSS received an unsupported document format, the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

```
urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError
```

and a `<dss:ResultMinor>` of

`urn:be:e-contract:dssp:1.0:resultminor:UnsupportedMimeType`

In case the DSS detected an error in one of the signatures , the DSS service returns in `<dss:SignResponse>` a `<dss:ResultMajor>` of

`urn:oasis:names:tc:dss:1.0:resultmajor:RequesterError`

and a `<dss:ResultMinor>` of

`urn:oasis:names:tc:dss:1.0:resultminor:invalid:IncorrectSignature`

5. Long-term validity of signatures

To discuss long-term validity of signatures, we first introduce the following notation:

Self-claimed signing time:

T_s

Time stamp time:

T_{tsa}

Now:

T_{now}

X509 certificate chain of signer:

$X509_s$

X509 certificate chain of time stamp authority:

$X509_{tsa}$

Set of signer certificate status information (can be a combination of CRL and OCSP), collected at time `t` :

$CRL_s(t)$

Set of TSA certificate status information (can be a combination of CRL and OCSP), collected at time `t` :

$CRL_{tsa}(t)$

PKI validation at time `t` (can be in the past) of certificate `X509` with revocation data `CRL` :

`validate(X509, CRL, t)`

Note that $validate(X509, CRL(t_1), t_2)$ will fail if t_2 is too far from t_1 , causing the revocation data $CRL(t_1)$ captured at time t_1 to be already expired at validation time t_2 .

For the signing certificate $X509_s$ it is clear that we want:

$validate(X509_s, CRL_s(T_s), T_s)$

If the signature contains a time stamp, we want

$T_{tsa} - T_s < max\_dt$

with `max_dt` some DSS application specific setting. We do this to ensure that PKI validations using either T_{tsa} or T_s yield the same result. This because T_s actually cannot be trusted.

For the validation of the time stamp certificate chain $X509_{tsa}$ however, we cannot use:

$validate(X509_{tsa}, CRL_{tsa}(T_{tsa}), T_{tsa})$

Imagine that the TSA gets hacked after time t with $t > T_{tsa}$. Then T_{tsa} cannot be trusted anymore. This means that for the validation of the TSA certificate chain we need to use:

$validate(X509_{tsa}, CRL_{tsa}(T_{now}), T_{now})$

Even when using $CRL_{tsa}(T_{now})$ it makes sense to store $CRL_{tsa}(T_{tsa})$ as part of the signature in case the CA of the TSA becomes unavailable due to a major security event, or if you receive a signature that skipped the time stamp renewal and thus for which $CRL_{tsa}(T_{now})$ is also not available anymore. This can happen in practice as CAs are not required to keep expired certificates on their CRLs. Another reason to store $CRL_{tsa}(T_{tsa})$ is because certificates can get suspended for a short period of time. To be able to check whether the $X509_{tsa}$ was suspended during creation of the time stamp at time T_{tsa}, you need to have $CRL_{tsa}(T_{tsa})$ available.

Right before the certificate of the TSA (or one of the intermediate certificates of the corresponding certificate chain) expires, you should capture CRL_{tsa} within the signature. Ideally, after TSA expiration, you should get the guarantee that the TSA private key is destroyed. However such guarantee cannot hold in reality. So right before expiration of the TSA certificate, we must do something else to secure this event.

Let us introduce a second time stamp authority. Hence we have an inner TSA used to create an inner time stamp:

$X509_{itsa}$

and an outer TSA used to create an outer time stamp:

$X509_{otsa}$

The outer TSA certificate chain should be checked via:

$validate(X509_{otsa}, CRL_{otsa}(T_{now}), T_{now})$

If this yields a valid $X509_{otsa}$, then T_{otsa} can be trusted.

The inner TSA certificate can now be checked via:

$validate(X509_{itsa}, CRL_{itsa}(T_{otsa}), T_{otsa})$

This means that when creating the outer time stamp (at time T_{otsa}), we need to capture the revocation data $CRL_{itsa}(T_{otsa})$ of the inner TSA certificate. Even when we already have $CRL_{itsa}(T_{itsa})$ as part of the signature we still need to capture $CRL_{itsa}(T_{otsa}$

) as $\mathtt{CRL}_{\mathtt{itsa}}(\mathtt{T}_{\mathtt{itsa}})$ might already contain expired revocation data (certificate status information) at time $\mathtt{T}_{\mathtt{otsa}}$ and thus:

$\mathtt{validate(X509}_{\mathtt{itsa}}\mathtt{, CRL}_{\mathtt{itsa}}\mathtt{(T}_{\mathtt{itsa}}\mathtt{), T}_{\mathtt{otsa}}\mathtt{)}$

would otherwise fail.

Again note that keeping track of $\mathtt{CLR}_{\mathtt{itsa}}(\mathtt{T}_{\mathtt{itsa}})$ still makes sense to be able to check whether $\mathtt{X509}_{\mathtt{itsa}}$ was not suspended during the creation of the inner time stamp at time $\mathtt{T}_{\mathtt{itsa}}$. Thus ideally we keep track of both $\mathtt{CRL}_{\mathtt{itsa}}(\mathtt{T}_{\mathtt{itsa}})$ and $\mathtt{CRL}_{\mathtt{itsa}}(\mathtt{T}_{\mathtt{otsa}})$.

This process of time stamp renewal reoccurs every time the outer TSA certificate chain expires.

This process also occurs when one of the used digest algorithms has been shown to be weak for verification purposes. In this case the outer time stamp should use a stronger digest algorithm.

The end result of a signature verification is:

$\mathtt{(X509}_{\mathtt{s}}\mathtt{, T}_{\mathtt{s}}\mathtt{)}$

The above reasoning is very much in favour of the time stamp and document notarization business. Let's approach it from another point of view. Suppose we follow the above time stamp renewal strategy. What does it eventually bring us?

It is clear that the inner most time stamp, called the signature time stamp, is only there to acknowledge the self-claimed signing time via:

$\mathtt{T}_{\mathtt{tsa}} - \mathtt{T}_{\mathtt{s}} < \mathtt{max\_dt}$

Suppose something happens with this time stamping authority TSA. The only meaningful reaction to such an event would be to redo the time stamp. However, we probably won't be able to maintain:

$\mathtt{T}_{\mathtt{tsa}} - \mathtt{T}_{\mathtt{s}} < \mathtt{max\_dt}$

And hence cannot assert the self-claimed signing time $\mathtt{T}_{\mathtt{s}}$ anymore. So what's to point in redoing this time stamp anyway?

Similar for the outer most time stamp authority `otsa` , we have that redoing an outer time stamp in case of a security event with its TSA, will make the following security check impossible:

$\mathtt{validate(X509}_{\mathtt{itsa}}\mathtt{, CRL}_{\mathtt{itsa}}\mathtt{(T}_{\mathtt{otsa}}\mathtt{), T}_{\mathtt{otsa}}\mathtt{)}$

Since we just witnessed a security event with the inner TSA `itsa` at time `t` with $\mathtt{t} > \mathtt{T}_{\mathtt{otsa}}$. So again, there is little reason in redoing this outer most time stamp.

The irony of it all is that the only possible strategy to cope with these events is to renew the time stamps even more frequently, and hence is even more in favour of the TSA and notarization business. Here we would also need to always create at least two time stamps instantly with zero correlation as it comes to security events. Something impossible to guarantee.

It is a fact that long-term validity of signatures is economically and practically unfeasible.

So right before the signature time stamp expires you simply capture

$CRL_{tsa}(T_{now})$

one last time and you end the sequence there. If you're certain that the CA of the TSA will never remove expired certificates from its CRLs, you even do not have to do anything.

Executive summary: if you lose a TSA, you're foobar anyway.

6. XML Schema

The XML schema for the Digital Signature Service protocol is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:be:e-contract:dssp:1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    xmlns:tns="urn:be:e-contract:dssp:1.0"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:wsa="http://www.w3.org/2005/08/addressing">

    <annotation>
        <documentation>
            Digital Signature Service Protocol XML schema.
            Copyright (C) 2013-2016 e-Contract.be BVBA.
        </documentation>
    </annotation>

    <import namespace="urn:oasis:names:tc:SAML:2.0:assertion"
        schemaLocation="saml-schema-assertion-2.0.xsd" />

    <import namespace="urn:oasis:names:tc:SAML:2.0:metadata"
        schemaLocation="saml-schema-metadata-2.0.xsd" />

    <import namespace="http://www.w3.org/2005/08/addressing"
        schemaLocation="ws-addr.xsd" />

    <element name="TimeStampRenewal" type="tns:DeadlineType" />

    <complexType name="DeadlineType">
        <annotation>
            <documentation>
                This complex type indicates when the timestamps
                on a signed document should be refreshed.
            </documentation>
        </annotation>
        <attribute name="Before" type="xs:dateTime" use="required" />
    </complexType>

    <element name="AttestationRequest" type="tns:AttestationRequestType" />

    <complexType name="AttestationRequestType">
        <annotation>
            <documentation>
                This complex type indicates that the web application wants
                to receive an attestation SAML assertion.
            </documentation>
        </annotation>
    </complexType>

    <element name="AttestationResponse" type="tns:AttestationResponseType" />

    <complexType name="AttestationResponseType">
        <annotation>
            <documentation>
                This complex type is used by the DSS to deliver the attestation
                SAML assertion as requested by the web application.
            </documentation>
        </annotation>
        <sequence>
            <element ref="saml2:Assertion" />
        </sequence>
    </complexType>

    <complexType name="EndpointType">
        <sequence>
            <element ref="wsa:EndpointReference"/>
        </sequence>
    </complexType>

    <complexType name="DigitalSignatureServiceDescriptorType">
        <annotation>
            <documentation>
                This complex type should be used within the DSS SAML 2.0
                metadata document.
            </documentation>
        </annotation>
        <complexContent>
            <extension base="md:RoleDescriptorType">
                <sequence>
                    <element name="WebServiceEndpoint" type="tns:EndpointType">
                        <annotation>
                            <documentation>
                                Indicates the location of the DSS SOAP
                                web service endpoint.
                            </documentation>
                        </annotation>
                    </element>
                    <element name="BrowserPostEndpoint" type="tns:EndpointType">
                        <annotation>
                            <documentation>
                                Indicates the location of the DSS browser post
                                endpoint.
                            </documentation>
                        </annotation>
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>

</schema>
```

A. Digital Signature Service Protocol Specifications License

B. Digital Signature Service Protocol Project License