Feedback on Local Signature Computation dating 18 April 2012 to Ernst Jan van Nigtevecht

Frank Cornelis
frank.cornelis@fedict.be
18 september 2012

page 7: “PKCS#1 signature (or should be allow 'raw' signature values)”.
What's the difference between PKCS#1 and 'raw' signatures? In my view PKCS#1 is the 'raw' signature. The lower-level aka textbook crypto is out of scope for obvious security reasons I guess.

Page 9: Figure 3.
I don't see the added value of (1) and (6). Why would we standardize those requests/responses? Nothing wrong with plain HTTP (over SSL).
(3) and (5) on the other hand is where the interesting stuff happens. The big problem here is how to initiate the communication with the local device. The DSS Server cannot just send over a DSS SignRequest to the user agent (which might be just a regular web browser not aware of any local device at all). So the DSS Server will require some out-of-band means to trigger the user agent into activating the local signature computation capabilities. This might for example be done by executing some Javascript within the web browser, or by loading some Java browser applet. Because web browsers are client-only (WebSocket didn't take off?) the local signature computation protocol run will be initiated by the web browser instead of the DSS Server (PAOS). Also, the communication will probably not be limited to a single SignRequest/SignResponse. For the construction of advanced electronic signatures, you first have to acquire the signatory's identity (X509 certificate chain) as this identity has to be part of the signature (AdES-BES). So there is some additional ping-pong between the DSS Server and the local device 'tranceiver' (plugin, or browser extension, ...).

Page 9: paragraph accompanying Figure 4.
In my view it is not the web application that provides the user interface of the signing process. Basically step (a) redirect the web browser from the web application to the DSS Server. Here the DSS Server takes over the user interface for the visualization/signing process. The DSS Server known how to communicate with the local device for the creation of an (advanced) electronic signature. The communication between the DSS Server and the local device could be using the OASIS DSS protocol (see comments on Figure 5 above), but this might as well be a proprietary protocol. At the end, the DSS Server redirects back to the web application which receives a signed document.

Example protocol run: web application to DSS Server and back.
I'll give a protocol run as way to detail on a typical use case (for the eID DSS that is).
The following examples already use optional inputs instead of wrapper elements.
The user agent is browsing some web application. At a certain point in time the web application wants the user to sign a document. The web application initiates a Browser POST towards the DSS Server by means of the following HTML page:

```
<html>
	<head><title>DSS Browser POST</title></head>
	<body>
		<p>Redirecting to the DSS Server...</p>
		<form method=”post” action=”https://dss.com/post”>
			<input type=”hidden” name=”SignRequest” value=”...”/>
		</form>
		<script type=”text/javascript”>
			window.onload = function() {
```

```
                        document.forms[0].submit();
                  }
            </script>
      </body>
</html>
```

Here the SignRequest of the HTML form contains the base64 encoded dss:SignRequest message. This dss:SignRequest message looks as follows:

```
<dss:SignRequest Profile="urn:oasis:names:tc:dss-x:1.0:profiles:localsig">
      <dss:OptionalInputs>
            <dss:SignaturePlacement WhichDocument="#doc1"
                  CreateEnvelopedSignature="true"/>
            <tns:SignRequestContext ID="requiredId"
                  IssueInstant="some dateTime"
                  Destination="web application landing page URL"
                  ProtocolBinding=
            "urn:oasis:names:tc:dss-x:1.0:profiles:localsig:bindings:HTTP-POST">
                  <ds:Signature>
                        <ds:SignedInfo>
                              <ds:CanonicalizationMethod Algorithm="..."/>
                              <ds:SignatureMethod Algorithm="..."/>
                              <ds:Reference URI="">
                                    <ds:Transforms>
                                          <ds:Transform Algorithm=
                        "http://www.w3.org/2000/09/xmldsig#enveloped-signature">
                                    </ds:Transforms>
                                    <ds:DigestMethod Algorithm="...."/>
                                    <ds:DigestValue>...</ds:DigestValue/>
                              </ds:Reference>
                        </ds:SignedInfo>
                        <ds:SignatureValue>...</ds:SignatureValue>
                        <ds:KeyInfo>...</ds:KeyInfo>
                  </ds:Signature>
            </tns:SignRequestContext>
      </dss:OptionalInputs>
      <dss:InputDocuments>
            <dss:Document ID="doc1">
                  <dss:Base64XML>the document base64 encoded</dss:Base64XML>
            </dss:Document>
      </dss:InputDocuments>
</dss:SignRequest>
```

Basically tns:SignRequestContext provides message-level security for the dss:SignRequest message. The web application signs the message with some key that is trusted by the DSS Server. Now the DSS Server takes over the user agent. After signing the document (in this use case, the actual signing between DSS Server and local device is using a proprietary protocol) 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 method="post" action="URL of SignRequest Destination attr">
                  <input type="hidden" name="SignResponse" value="..."/>
            </form>
            <script type="text/javascript">
                  window.onload = function() {
                        document.forms[0].submit();
                  }
            </script>
      </body>
</html>
```

Where SignResponse of the HTML form contains the base64 encoded dss:SignResponse message. This dss:SignResponse message looks as follows:

```
<dss:SignResponse Profile=”urn:oasis:names:tc:dss-x:1.0:profiles:localsig”>
      <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:Base64XML>the signed document</dss:Base64XML>
                  </dss:Document>
            </dss:DocumentWithSignature>
            <tns:SignResponseContext ID=”requiredId2”
                  InResponseTo=”previous requiredId request value”
                  IssueInstance=”...”
                  Destination=”...”>
                  <ds:Signature>
                        <ds:SignedInfo>
                              <ds:CanonicalizationMethod Algorithm=”...”/>
                              <ds:SignatureMethod Algorithm=”...”/>
                              <ds:Reference URI=””>
                                    <ds:Transforms>
                                          <ds:Transform Algorithm=
                        ”http://www.w3.org/2000/09/xmldsig#enveloped-signature”>
                                    </ds:Transforms>
                                    <ds:DigestMethod Algorithm=”....”/>
                                    <ds:DigestValue>...</ds:DigestValue/>
                              </ds:Reference>
                        </ds:SignedInfo>
                        <ds:SignatureValue>...</ds:SignatureValue>
                        <ds:KeyInfo>...</ds:KeyInfo>
                  </ds:Signature>
            </tns:SignResponseContext>
      </dss:OptionalOutputs>
      <dss:SignatureObject>
            <dss:SignaturePtr WhichDocument=”#doc1”/>
      </dss:SignatureObject>
</dss:SignResponse>
```

The DSS Server signs the message with some key that is trusted by the web application.

Example protocol run: artifact binding + secure conversation

In this example we'll again have a sign request and sign response via a web browser POST. However this is preceded by a SOAP call to transfer the document and to setup a secure conversation.

First the web application executes a SOAP request to the DSS Server:

```
<soap:Envelope>
      <soap:Body>
            <dss:SignRequest Profile=
                  ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:artifact”>
                  <dss:OptionalInputs>
                        <dss:AdditionalProfile>
                  urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing
                        </dss:AdditionalProfile>
                        <dss:AdditionalProfile>
                  urn:oasis:names:tc:dss-x:1.0:profiles:localsig:secure-conv
                        </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:OptionalInputs>
                  <dss:InputDocuments>
                        <dss:Document ID=”doc1”>
                              <dss:Base64XML>the document</dss:Base64XML>
                        </dss:Document>
                  </dss:InputDocuments>
            </dss:SignRequest>
      </soap:Body>
</soap:Envelope>
```

The artifact binding allows for transportation of the document over SOAP, which is faster than via a web browser POST. Using a secure conversation also greatly simplifies the trust setup between web application and DSS Server, which can default to basic SSL.

The DSS Server responds as follows:

```
<soap:Envelope>
      <soap:Body>
            <dss:SignResponse Profile=
                  ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:artifact”>
                  <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>
                                                <wsc:Identifier>
                                                      token-id
                                                </wsc:Identifier>
                                          </wsc:SecurityContextToken>
                                    </wst:RequestedSecurityToken>
                                    <wst:RequestedUnattachedReference>
                                          <wsse:SecurityTokenReference>
                                                <wsse:Reference
ValueType="http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/sct"
                        URI="token-id" />
                                          </wsse:SecurityTokenReference>
                                    </wst:RequestedUnattachedReference>
                                    <wst:RequestedProofToken>
                                          <wst:ComputedKey>
                  http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1
                                          </wst:ComputedKey>
                                    </wst:RequestedProofToken>
                                    <wst:Entropy>
                                          <wst:BinarySecret Type=
```

```
                  "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Nonce">
                                          ...
                                    </wst:BinarySecret>
                              </wst:Entropy>
                              <wst:KeySize>256</wst:KeySize>
                              <wst:Lifetime>
                                    <wsu:Created>...</wsu:Created>
                                    <wsu:Expires>...</wsu:Expires>
                              </wst:Lifetime>
                        </wst:RequestSecurityTokenResponse>
                  </wst:RequestSecurityTokenResponseCollection>
            </dss:OptionalOutputs>
      </dss:SignResponse>
</soap:Body>
</soap:Envelope>
```

Via the responseID the web application can further reference the uploaded document that has to be signed. The secure conversation token and corresponding proof-of-possession key is to be used to secure the Browser POST messages between web application and DSS Server.
Next the web application initiates a Browser POST towards the DSS Server by means of the following HTML page:

```
<html>
      <head><title>DSS Browser POST</title></head>
      <body>
            <p>Redirecting to the DSS Server...</p>
            <form method=”post” action=”https://dss.com/post”>
                  <input type=”hidden” name=”PendingRequest” value=”...”/>
            </form>
            <script type=”text/javascript”>
                  window.onload = function() {
                        document.forms[0].submit();
                  }
            </script>
      </body>
</html>
```

Here the PendingRequest of the HTML form contains the base64 encoded async:PendingRequest message. This async:PendingRequest message looks as follows:

```
<async:PendingRequest Profile=
                  ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:artifact”>
      <dss:OptionalInputs>
            <dss:AdditionalProfile>
                  urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing
            </dss:AdditionalProfile>
            <async:ResponseID>responseId</async:ResponseID>
            <dss:SignaturePlacement WhichDocument=”#doc1”
                  CreateEnvelopedSignature=”true”/>
            <tns:PendingRequestContext ID=”requiredId”
                  IssueInstant=”some dateTime”
                  Destination=”web application landing page URL”
                  ProtocolBinding=
            ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:bindings:HTTP-POST”>
                  <ds:Signature>
                        <ds:SignedInfo>
                              <ds:CanonicalizationMethod Algorithm=”...”/>
                              <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:Transforms>
                                    <ds:DigestMethod Algorithm=”....”/>
                                    <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>
            </tns:PendingRequestContext>
      </dss:OptionalInputs>
</async:PendingRequest>
```

Via the responseID the web application references the document that was previously transmitted to the DSS Server via the SOAP call. The message-level signature is using the security token's corresponding proof-of-possession key.

After signing the document (in this use case, the actual signing between DSS Server and local device is using a proprietary protocol) 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 method=”post” action=”URL of PendingRequest Destination attr”>
                  <input type=”hidden” name=”SignResponse” value=”...”/>
            </form>
            <script type=”text/javascript”>
                  window.onload = function() {
                        document.forms[0].submit();
                  }
            </script>
      </body>
</html>
```

Where SignResponse of the HTML form contains the base64 encoded dss:SignResponse message. This dss:SignResponse message looks as follows:

```
<dss:SignResponse Profile=
                  ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:artifact”>
      <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>
            <tns:SignResponseContext ID=”requiredId2”
                  InResponseTo=”previous requiredId request value”
                  IssueInstance=”...”
                  Destination=”...”>
                  <ds:Signature>
                        <ds:SignedInfo>
                              <ds:CanonicalizationMethod Algorithm=”...”/>
                              <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:Transforms>
                                    <ds:DigestMethod Algorithm=”....”/>
                                    <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>
            </tns:SignResponseContext>
      </dss:OptionalOutputs>
</dss:SignResponse>
```

The DSS Server signs the message with the secure conversation token's proof-of-possession key. Finally the web application can request the signed document from the DSS Server via a SOAP call:

```
<soap:Envelope>
      <soap:Body>
            <async:PendingRequest Profile=
                  ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:artifact”>
                  <dss:OptionalInputs>
                        <dss:AdditionalProfile>
                  urn:oasis:names:tc:dss:1.0:profiles:asynchronousprocessing
                        </dss:AdditionalProfile>
                        <async:ResponseID>responseId</async:ResponseID>
                  </dss:OptionalInputs>
            </async:PendingRequest>
      </soap:Body>
</soap:Envelope>
```

And the DSS Server returns the signed document:

```
<soap:Envelope>
      <soap:Body>
            <dss:SignResponse Profile=
                  ”urn:oasis:names:tc:dss-x:1.0:profiles:localsig:artifact”>
                  <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:Base64XML>
                                          the signed document
                                    </dss:Base64XML>
                              </dss:Document>
                        </dss:DocumentWithSignature>
                  </dss:OptionalOutputs>
                  <dss:SignatureObject>
                        <dss:SignaturePtr WhichDocument=”#doc1”/>
                  </dss:SignatureObject>
            </dss:SignResponse>
      </soap:Body>
</soap:Envelope>
```