DIGITALISERINGSSTYRELSEN

NemLog-in

NL2 Signing Service Interface

Contents

Changelog

Date	Version	Change description	Initials
2012-06-08	1.0	Initial version	CVSH
2012-06-11	1.1	Added chapter about requirements to service providers	TG
2013-05-27	1.2	Updated document to reflect that the component deployed in the integration test environment is no longer a stub. SignatureProofID has been changed to CA1 and CA2.	CVSH
2013-05-30	1.3	Reviewed with comments	ASEP
2013-05-30	1.4	Comments incorporated in document	CVSH
2013-06-13	1.5	Added section about the signing web service Added section about the back-channel web service	CVSH
2013-06-21	1.6	Corrected processing description for SignedFingerprint in Web Service.	ASEP
2013-06-24	1.7	Added code snippet example for processing request parameters	ASEP
2021-12-13	1.8	Updated code snippet in section 3.1 to handle OCES3 certificate private keys.	MDBEC

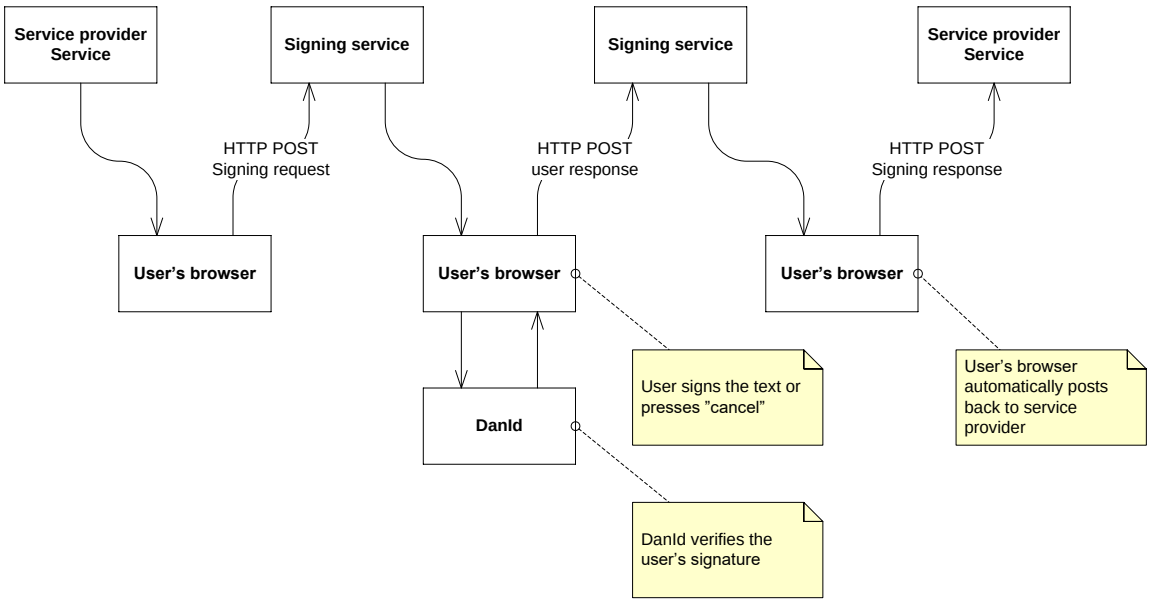
1 Purpose

The purpose of the signing service in the integration test environment is to provide a way for service providers to develop and test services that use the signing service.

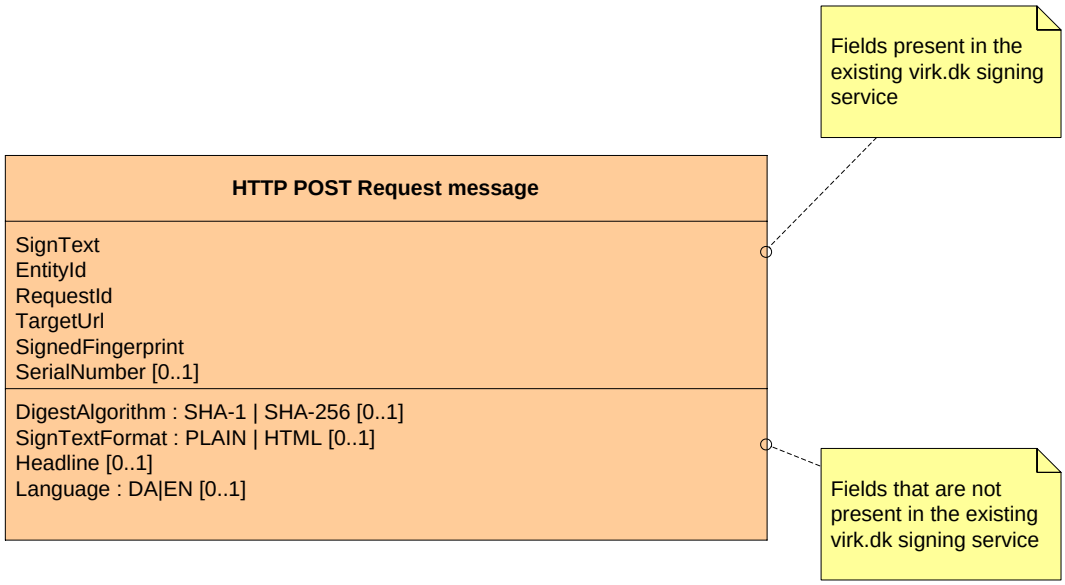
Service providers that use the existing signing service provided by virk.dk should be aware that the interface has changed slightly. The signing service in the integration test environment will allow these service providers to implement and test the changes necessary in order to use the new signing service.

2 Overview

The signing service implements the following workflow using HTTP POST-based interfaces:



3 Signing Request Message



The signing service is invoked by sending an HTML POST message, containing the fields shown above. How this is done is up to the service provider, but one possible solution is to send an html page to the end user containing a JavaScript that automatically posts the page to the signing service:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Verificer signatur</title>
  <script type="text/javascript">
    function AutoPost()
    {
      document.forms["Sign"].submit();
    }
  </script>
</head>
<body onload="AutoPost()">
  <form id="Sign" runat="server" method="post" action="https://sign...in.dk">
    <input name="SignText" id="SignText" type="hidden" />
    <input name="EntityId" id="EntityId" type="hidden" />
    <input name="RequestId" id="RequestId" type="hidden" />
    <input name="TargetUrl" id="TargetUrl" type="hidden" />
    <input name="SignedFingerprint" id="SignedFingerprint" type="hidden" />
  </form>
</body>
</html>
```

The signing request message consists of the following fields:

Field name	Virk.dk compatible	Description	Mandatory	Processing
SignText	Yes	This is the actual agreement text that the end user needs to sign. The text may be plain text or applet html.	Yes	SignText = Base64Encode(Utf8Encode([agreement text])))

Field name	Virk.dk compatible	Description	Mandatory	Processing
SignTextFormat	New	This field indicates the type of the SignText. Possible values are PLAIN or HTML. Note that the DanId applet also supports XML and NetId. These are **not** supported by the signing service.	No	If the SignTextFormat field is missing the value defaults to PLAIN. HTML is only allowed if CertificateType is set to OCES2.
EntityId	Yes	Each service provider has a unique EntityId which is used to identify the service provider in the federation.	Yes	
RequestId	Yes	Unique identifier (just a text, not necessarily a GUID) that the service provider uses to match the response from the signing service with the request.	Yes	
TargetUrl	Yes	The URL that the signing service should use when sending the response to the service provider. Note that the signing service should always use https when sending the response to the service provider, even if http was specified in the TargetUrl field.	Yes	Replace(TargetUrl, “http”, “https”)
SignedFingerPrint	No	A signed digest of the fields SignText, EntityId and TargetUrl. The digest must be signed using the service provider’s private key. The hashing algorithm can be either SHA256 or SHA1 depending on the value of the DigestAlgorithm field. The hashing algorithm defaults to SHA1 if the DigestAlgorithm is not specified.	Yes	SignedFingerPrint = Base64Encode(SignSha256WithRsa(Utf8Encode(Concat(SignText, EntityId, TargetUrl))))

Field name	Virk.dk compatible	Description	Mandatory	Processing
SerialNumber	Yes	Optional field indicating the serial number on the certificate that the user needs to use for signing the agreement	No	SerialNumber = Base64Encode(Base10String([certificate's serial number]))
Headline	New	Optional text describing the context of the signing process that will be displayed in the signing web page header	No	Base64Encode([headline text])
Language	New	Optional parameter indicating the language that the signing page should be displayed in. Possible values are EN (English) and DA (Danish).	No	If the Language field is missing, the language defaults to DA.
DigestAlgorithm	New	Optional parameter indicating the hash algorithm that was used when creating the SignedFingerprint. Allowed values are: SHA-1 and SHA-256.	No	If the DigestAlgorithm field is missing, the algorithm defaults to SHA-1.

3.1 Processing Request Parameters

This section contains an example on how to process the request parameters for invoking the Signing Service. The example is implemented in C#/.Net 4.0 using the crypto API in the System.Security.Cryptography namespace.

```
// Input
var entityId = "https://example.com";
var signText = "Text to be signed";
var signTextB64 = Convert.ToBase64String(Encoding.UTF8.GetBytes(signText));
var targetUrl = "https://www.example.com/ProcessSigningResponse.aspx";
var certificate = new X509Certificate2(Resource.TestCertificate, "PassWord");
var hashAlgorithm = "SHA256";

// Generate digest
var digest = string.Concat(signTextB64, entityId, targetUrl);
var digestBytes = Encoding.UTF8.GetBytes(digest);

// Sign digest
var key = certificate.GetRSAPrivateKey();
var hashAlgo = HashAlgorithmName.SHA256;
var signedFingerprint = key.SignData(digestBytes, hashAlgo, RSASignaturePadding.Pkcs1);
var signedFingerprintB64 = Convert.ToBase64String(signedFingerprint);

// Output
Console.WriteLine("SignText: {0}", signTextB64);
Console.WriteLine("EntityId: {0}", entityId);
Console.WriteLine("TargetUrl: {0}", targetUrl);
Console.WriteLine("SignedFingerprint: {0}", signedFingerprintB64);
Console.WriteLine("DigestAlgorithm: {0}", hashAlgorithm);
```

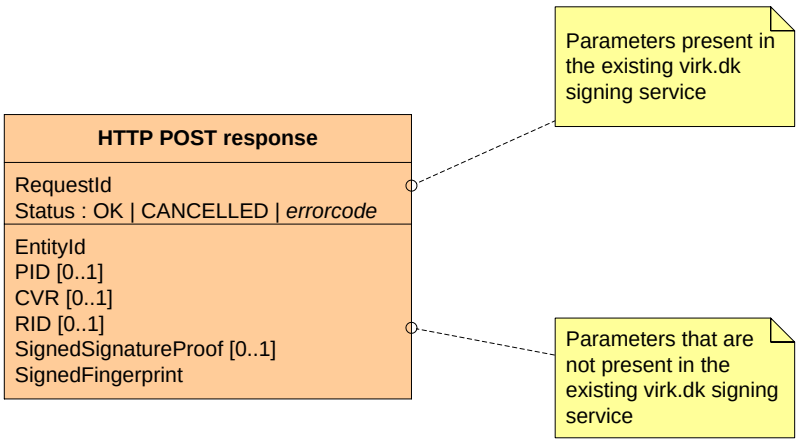
In this example, the certificate is retrieved from a resource, which is of course not advisable – use the certificate store.

Note that SHA256 is used for hashing the digest, which is not the default algorithm, consequently the DigestAlgortihm must be specified to Signing Service.

The example generates the following output, which can be passed to Signing Service using the HTML form earlier in section 3:

```
SignText: VGV4dCB0byBiZSBzaWduZWQ=

EntityId: https://example.com

TargetUrl: https://www.example.com/ProcessSigningResponse.aspx

SignedFingerprint:
OXg4B/Xt+ejfIQceiKY5xuP20EnQGhcSfFHI/qiec1AicAK6Lr6x7+qLn/5lh4mNOPp+wvfmkQunck4rUMD9yo3GC44HByZriHs
930/q44PwOw0zfGv3HjIqVp6hBZnEZ1jKag5FJsabNt2F/waqHKPTEcujg4LHZMqgBiIPqAw=

DigestAlgorithm: SHA256
```

4 Signing Response Message



After the signing service has been called with a signing request message, it will go through a process involving several steps, where the user is asked to sign the sign text. The process ends up with the user either signing the sign text or cancelling the signing request. Once this is done, the signing service immediately responds with a HTML POST to the target URL specified by the service provider.

The signing request message will be sent from the end user's browser by providing a JavaScript that automatically posts the page to the target URL.

The signing response message contains the following fields:

Field name	Description	Mandatory	Processing
RequestId	Unique identifier (just a text, not necessarily a GUID) that the service provider uses to match the response from the signing service with the request.	Yes	The signing service returns the same RequestId that was provided in the *signing request message*.
Status	Indicates the result of the signing operation: OK: The user signed the sign text. CANCELLED: The user pressed the cancel button. *ErrorCode:* *An error occurred. The error code indicates the type of error.*	Yes	Error codes are described in section 8 below.
EntityId	Each service provider has a unique EntityId, which is used to identify the service provider in the federation.	Yes	The signing service returns the same EntityId that was provided in the *signing request message*.
PID	In case a citizen certificate was used for signing, this field contains the PID of the certificate used.	No	

Field name	Description	Mandatory	Processing
CVR	In case an employee certificate was used for signing, this field contains the CVR number of the certificate used.	No	
RID	In case an employee certificate was used for signing, this field contains the RID of the certificate used.	No	
SignedSignatureProof	The signed document received by DanId and re-signed by the signing service.	No	Encoded according to: Base64Encode(Utf8Encode(SignedSignatureProof))
SignedFingerPrint	A signed digest of the fields RequestId, Status, EntityId, PID, CVR, RID and SignedSignatureProof. The digest is signed with the signing service's certificate.	Yes	SignedFingerPrint = Base64Encode(SignSha256WithRsa(Utf8Encode(Concat(RequestId, Status, EntityId, PID, CVR, RID, Base64Encode(Utf8Encode(SignedSignatureProof))))))

5 Back Channel Web Service

In the virk.dk signing service implementation, there was a back-channel web service that allowed service providers to retrieve the signature proof. Please note that this web service does **not** exist in the current version of the signing service. Instead, the signed signature proof is returned to the service provider as part of the signing response message. The service provider is expected to save the sign text as well as the CA1 and CA2 values of the signed signature proof (see section 6 below). The signing service does **not** store a complete copy of the signed signature proof. However, if the service provider is able to present the sign text as well as the CA1 and CA2 values, it will be possible to reconstruct the signed signature proof from information stored by the signing service.

6 SignedSignatureProof

The SignedSignatureProof proofs the SignText was signed using a specific certificate. The SignedSignatureProof is signed by the end user and the re-signed by the signing service. This means that the SignedSignatureProof contains two nested XMLDSig structures, as shown in the following example.

```
<?xml version="1.0" encoding="utf-16"?>
<SignatureProof Timestamp="2012-10-22T08:47:46.1565821+02:00"
                xmlns="uri://dk.digst.nemlogin.signingservice"
                CA1="AMB5QUQVA0RKQe9ACGEBNP3L25DG417A3QDL2OOO2LT6HRRDI9A6H"
                CA2="6DNPG83TV903QeDKND2EFH7TVQUG417A3QFQ4K0D6ED59V02FEJC5">
  <SigningAppletOutput Id="SignedByKFOBS">
    <openoces:signature xmlns:openoces="http://www.openoces.org/2006/07/signature#" version="0.1">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                       xmlns:openoces="http://www.openoces.org/2006/07/signature#">
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256">
          </ds:SignatureMethod>
          <ds:Reference URI="#ToBeSigned">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"></ds:DigestMethod>
            <ds:DigestValue>pwKD859Jxn0LWaaH+VMY3MkKu1lZMOkawsE5DZsGt9E=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
          jb93i/Rc2ElJIZaz5or2umcSxQclSHlb9jfhCqbrDB3+GARyTl6t4modC2batr3OtTpUqCQrozd8
          KXux9K5YMRHRhkGXNJAKNwcJmkCArBZrCb//A4WutxvD22eWk/nGCOTh/c1iqBZfQ66eh0CSK4ma
          6wseGJ8dakwjQ6avTmRxGPusY9BKwKzc6F1aw1b1bHewSXyksUsx2GV7v84payC/xLXaOJ4xMfWo
          RR2/5wj+Af9EWwrBA7phgTyMNC39RSTTFpvleHB3jDNoZoJ6KJ1zPqK+YEYf2bxNJI5CKvzJY/DT
          6+iB2rDVtEEjPLlHjn7ngS2nUnqNobmXhyNBZQ==
        </ds:SignatureValue>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>
              MIIGITCCBQmgAwIBAgIETAV29zANBgkqhkiG9w0BAQsFADBIMQswCQYDVQQGEwJESzESMBAGA1UE
              CgwJVFJVU1QyNDA4MSUwIwYDVQQDDBxUUlVTVDI0MDggU3lzdGVtdGVzdCBWSUlJIENBMB4XDTEy
              MDExNzE0MjUwNVoXDTE2MDExNzE0MjQzOFowdzELMAkGA1UEBhMCREsxLjAsBgNVBAoMJUU1JQ1JP
              U09GVCBEQU5NQVJLIEFwUyAvLyBDVlI6MTM2MTI4NzAxODAUBgNVBAMMDU1pa2UgU29mdXNzZW4w
              IAYDVQQFExlDVlI6MTM2MTI4NzAtUklEOjMxNDQ5MjQxMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
              MIIBCgKCAQEAn6DuA5lQC7r9j7UgGcncSxC1b7KDzorI8y97RO8xIi3jP6FfB7e/5hKuCk8pTP2t
              6vhsoVFcoColIWgsKadwlX5Uvj3tl4M7SuLygvxgHEL4uXGzKH6ISgppP/WYwHjVyfU1iw++X/ec
              pwBOhS8SpLJqrFAdK20Ej+aXGnfPJG8yFeHzX+wgg63CaV9RcE71XZqR/+h6+XCkCPHDzmFULAeX
              jo5QEnh7WNJpDHWx4JhkspXAnf85aWPrm59lY5RozXMswLVm3lqVTF7CbtZCQUDnqraj9lo97DcK
              N4yxDiJU3ZKFf+O8zrmZHVxssDxYnM5/+tqK8MYMtJ8HV90PcQIDAQABo4IC4jCCAt4wDgYDVR0P
              AQH/BAQDAgP4MIGUBggrBgEFBQcBAQSBhzCBhDA7BggrBgEFBQcwAYYvaHR0cDovL29jc3Auc3lz
              dGVtdGVzdDgudHJ1c3QyNDA4LmNvbS9yZXNwb25kZXIwRQYIKwYBBQUHMAKGOWh0dHA6Ly9tLmFp
              YS5zeXN0ZW10ZXN0OC50cnVzdDI0MDguY29tL3N5c3RlbXRlc3Q4LWNhLmNlcjCCASAGA1UdIASC
              ARcwggETMIIBDwYNKwYBBAGB9FECBAYCBTCB/TAvBggrBgEFBQcCARYjaHR0cDovL3d3dy50cnVz
              dDI0MDguY29tL3JlcG9zaXRvcnkwgckGCCsGAQUFBwICMIG8MAwWBURhbklEMAMCAQEagatEYW5J
              RCB0ZXN0IGNlcnRpZmlrYXRlciBmcmEgZGVubmUgQ0EgdWRzdGVkZXMgdW5kZXIgT0lEIDEuMy42
              LjEuNC4xLjMxMzEzLjIuNC42LjIuNS4gRGFuSUQgdGVzdCBjZXJ0aWZpY2F0ZXMgZnJvbSB0aGlz
              IENBIGFyZSBpc3N1ZWQgdW5kZXIgT0lEIDEuMy42LjEuNC4xLjMxMzEzLjIuNC42LjIuNS4wGAYD
              VR0RBBEwD4ENamVsZkBubml0LmNvbTCBqwYDVR0fBIGjMIGgMDqgOKA2hjRodHRwOi8vY3JsLnN5
              c3RlbXRlc3Q4LnRydXN0MjQwOC5jb20vc3lzdGVtdGVzdDguY3JsMGKgYKBepFwwWjELMAkGA1UE
              BhMCREsxEjAQBgNVBAoMCVRSVVNUMjQwODElMCMGA1UEAwwcVFJVU1QyNDA4IFN5c3RlbXRlc3Qg
              VklJSSBDQTEQMA4GA1UEAwwHQ1JMMTEyNDAfBgNVHSMEGDAWgBSWGzYTOyIpwj355/mT68bLPhJf
              BDAdBgNVHQ4EFgQUSytiMhYU0lqAesaUAYct9dPSW8cwCQYDVR0TBAIwADANBgkqhkiG9w0BAQsF
              AAOCAQEAFIrjc6k/48x90OoOSMXglNaSHdS2SF9kZ1o1oDF6+X09ktejRbPBjsqc/32tMOjFpHqj
              4kQy5kVFkrMD3T7tSUgEg50oXzVzD0j1Fyaw5mfgzroFbUQZpUpR63uy1Jsb0mdWkVULcWaA/De0
              18cngPIn0VAahAcHNSo3CLJT3VQaNCv9XvqN6nBYYIyn1pco0uGesaOFBGUg9FI5w6wTowRGaAlj
              xjcAd3orWHDZ+1zuZCFmrI9PrjGhqdrPVwrnYeS9DRGDwZ5z0/vF6MXqeNcimjEP0xfwT48SHsw+
              4U/R8i1bGaKEw/jbm7W9W24e4/DzDXfOglDjzet3GxoHlg==
            </ds:X509Certificate>
          </ds:X509Data>
          <ds:X509Data>
            <ds:X509Certificate>
              MIIFQTCCAymgAwIBAgIES+pu4zANBgkqhkiG9w0BAQsFADBPMQswCQYDVQQGEwJESzESMBAGA1UE
              ChMJVFJVU1QyNDA4MSwwKgYDVQQDEyNUUlVTVDI0MDggU3lzdGVtdGVzdCBWSUkgUHJpbWFyeSBD
              QTAeFw0xMDA1MTIxMjQzMjFaFw0yNTA1MTIxMzEzMjFaMEgxCzAJBgNVBAYTAkRLMRIwEAYDVQQK
              DAlUUlVTVDI0MDgxJTAjBgNVBAMMHFRSVVNUMjQwOCBTeXN0ZW10ZXN0IFZJSUkgQ0EwggEiMA0G
              CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDhVT0GVUM/Xt1CY0Asa0DZ+4OzhVWixuYdak/VxuoP
              BUBErrCjKBDELEwdAwr8NlXoqI6JZqrFzsq8+Pj9NuhmiAjS1tzwAIR1a6VaF+PUf8tsdofBaEjB
```

```
            U/Y3Oqtvnj1pcCHADlwqdVQeCE1ehPn1Jmk2ZtfDLRrOpAb0pbaQuFXeoBqlbRiyFRu+l16E6RFR
            wReqgMvcNcbB52G+N5W9/C4DewgV7NK3Lf9llp+7guHYWe2fqjUtGvqW0B29o9kBv7wuGAwZzWZ8
            TkW0Ph3Kf58Eq/JBdmwRx5S00c9yAN8IamUOyrvkKUK/9NSdF5szBkqjPVJ/93+ocX6jD6PjAgMB
            AAGjggEqMIIBJjAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwIBBjARBgNVHSAECjAIMAYG
            BFUdIAAwga8GA1UdHwSBpzCBpDA6oDigNoY0aHR0cDovL2NybC5zeXN0ZW10ZXN0Ny50cnVzdDI0
            MDguY29tL3N5c3RlbXRlc3Q3LmNybDBmoGSgYqRgMF4xCzAJBgNVBAYTAkRLMRIwEAYDVQQKEwlU
            UlVTVDI0MDgxLDAqBgNVBAMTI1RSVVNUMjQwOCBTeXN0ZW10ZXN0IFZJSSBQcmltYXJ5IENBMQ0w
            CwYDVQQDEwRDUkwxMB8GA1UdIwQYMBaAFCO6TDGQ4dPM0xuzG1q2yEXN04OXMB0GA1UdDgQWBBSW
            GzYTOyIpwj355/mT68bLPhJfBDANBgkqhkiG9w0BAQsFAAOCAgEAQM4OBCWowO58HU0Ak7FjEbde
            IKKXWi6jPg7KCXWrHSFruFmtjYiz41hO2yaMNmSJNBrW4u5wkqu7ZcbCaIPoOQVrTPZtqM/npEt7
            8J1bBZr1s4lMhzr7sg7HEYFYvsBaLF70hNpi4JSQjE4x9WwfpzwU7YzWOOm7UjRhybUs911/3cvp
            Lw7Txh8rntsAiE7IlAjJa4Tq6hH03bStUiY7zTHywE8/DPpfXuC3Mh4pZfYz/hfFvPdZTJL/yW1t
            PnmqoUi6KH2g8n58JJfZVwTGrL7H3LF5zsBZCWnZR7XdhrqoRR5HnFT+uI6IChZwhzOZVT69ukJJ
            KuApDgoAaoz0qEXh90BeVKwm8X1O9b1fwrfcEYD0g53+JCgtyv30hHX03dObAj+TQjNpZy9HvQNW
            LNe7fp+59dOCh87FJGlWfZGGbC3LblD6tctEfN6I+kqnFqPpjE2xAtnsYW8YClPlLuvb43+6MlWO
            R2MxTr/UyhOyrqmM8O1aEKU7jDVQDxS+E7kzBech9jvQOYB/WibU5vyGhIIWfZZmgUwN16iwOl2V
            Jx/N3QPxoORwSkDKKiwjOFx6IecFIJJnI6B1CA75scHXXYtKmWZm6UjNsdiyC4pzQ7fKOc1LpOx8
            D7+aXpsDl2HeB7MrHV1GBKJJPaQUujK3dgS+UDk0yFKcWMdXEFI=
          </ds:X509Certificate>
        </ds:X509Data>
        <ds:X509Data>
          <ds:X509Certificate>
            MIIGSDCCBDCgAwIBAgIES+pulDANBgkqhkiG9w0BAQsFADBPMQswCQYDVQQGEwJESzESMBAGA1UE
            ChMJVFJVU1QyNDA4MSwwKgYDVQQDEyNUUlVTVDI0MDggU3lzdGVtdGVzdCBWSUkgUHJpbWFyeSBD
            QTAeFw0xMDA1MTIwODMyMTRaFw0zNzAxMTIwOTAyMTRaME8xCzAJBgNVBAYTAkRLMRIwEAYDVQQK
            EwlUUlVTVDI0MDgxLDAqBgNVBAMTI1RSVVNUMjQwOCBTeXN0ZW10ZXN0IFZJSSBQcmltYXJ5IENB
            MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEApuuMpdHu/lXhQ+9TyecthOxrg5hPgxlK
            1rpjsyBNDEmOEpmOlK8ghyZ7MnSF3ffsiY+0jA51p+AQfYYuarGgUQVO+VM6E3VUdDpgWEksetCY
            Y8L7UrpyDeYx9oywT7E+YXH0vCoug5F9vBPnky7PlfVNaXPfgjh1+66mlUD9sV3fiTjDL12GkwOL
            t35S5BkcqAEYc37HT69N88QugxtaRl8eFBRumj1Mw0LBxCwl21GdVY4EjqH1Us7YtRMRJ2nEFTCR
            WHzm2ryf7BGd80YmtJeL6Roiidwl IgzvhoFhv4XdLHwzaQbdb9s141q2s9KDPZCGcgIgeXZdqY1V
            z7UBCMiBDG7q2S2ni7wpUMBye+iYVkvJD32srGCzpWqG7203cLyZCjq2oWuLkL807/Sk4sYleMA4
            YFqsazIfV+M0OVrJCCCkPysS10n/+ioleM0hnoxQiupujIGPcJMA8anqWueGIaKNZFA/m1IKwnn0
            CTkEm2aGTTEwpzb0+dCATlLyv6Ss3w+D7pqWCXsAVAZmD4pncX+/ASRZQd3oSvNQxUQr8EoxEULx
            Sae0CPRyGwQwswGpqmGm8kNPHjIC5ks2mzHZAMyTz3zoU3h/QW2T2U2+pZjUeMjYhyrReWRbOIBC
            izoOaoaNcSnPGUEohGUyLPTbZLpWsm3vjbyk7yvPqoUCAwEAAaOCASowggEmMA8GA1UdEwEB/wQF
            MAMBAf8wDgYDVR0PAQH/BAQDAgEGMBEGA1UdIAQKMAgwBgYEVR0gADCBrwYDVR0fBIGnMIGkMDqg
            OKA2hjRodHRwOi8vY3JsLnN5c3RlbXRlc3Q3LnRydXN0MjQwOC5jb20vc3lzdGVtdGVzdDcuY3Js
            MGagZKBipGAwXjELMAkGA1UEBhMCREsxEjAQBgNVBAoTCVRSVVNUMjQwODEsMCoGA1UEAxMjVFJV
            U1QyNDA4IFN5c3RlbXRlc3QgVklJIFByaW1hcnkgQ0ExDTALBgNVBAMTBENSTDEwHwYDVR0jBBgw
            FoAUI7pMMZDh08zTG7MbWrbIRc3Tg5cwHQYDVR0OBBYEFCO6TDGQ4dPM0xuzG1q2yEXN04OXMA0G
            CSqGSIb3DQEBCwUAA4ICAQCRJ9TM7sISJBHQwN8xdey4rxA0qT7NZdKICcIxyIC82HIOGAouKb3o
            HjIoMgxIUhA3xbU3Putr4+Smnc1Ldrw8AofLGlFYG2ypg3cpF9pdHrVdh8QiERozLwfNPDgVeCAn
            jKPNt8mu0FWBS32tiVM5DEOUwDpoDDRF27Ku9qTFH4IYg90wLHfLi+nqc2HwVBUgDt3tXU6zK4pz
            M0CpbrbOXPJOYHMvaw/4Em2r0PZD+QOagcecxPMWI65t2h/USbyO/ah3VKnBWDkPsMKjj5jEbBVR
            nGZdv5rcJb0cHqQ802eztziA4HTbSzBE4oRaVCrhXg/g6Jj8/tZlgxRI0JGgAX2dvWQyP4xhbxLN
            CVXPdvRV0g0ehKvhom1FGjIz975/DMavkybh0gzygq4sY9Fykl4oT4rDkDvZLYIxS4u1BrUJJJaD
            zHCeXmZqOhx8She+Fj9YwVVRGfxT4FL0Qd3WAtaCVyhSQ6SkZgrPvzAmxOUruI6XhEhYGlP5O8WF
            ETiATxuZAJNuKMJtibfRhMNsQ+TVv/ZPr5Swe+3DIQtmt1MIlGlTn4k40z4s6gDGKiFwAYXjd/kI
            D32R/hJPE41o9+3nd8aHZhBy2lF0jKAmr5a6Lbhg2O7zjGq7mQ3MceNeebuWXD44AxIinryzhqnE
            WI+BxdlFaia3U7o2+HYdHw==
          </ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                 xmlns:openoces="http://www.openoces.org/2006/07/signature#" Id="ToBeSigned">
        <ds:SignatureProperties>
          <ds:SignatureProperty>
            <openoces:Name>signtext</openoces:Name>
            <openoces:Value Encoding="base64" VisibleToSigner="yes">
              TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQsIGNvbnNlY3RldHVyIGFkaXBpc2NpbmcgZWxpdC
              4gUGhhc2VsbHVzIGN1cnN1cyBwdXJ1cyBuZWMgYXJjdSB0aW5jaWR1bnQgbGFvcmVldC4gTmFt
              IHBoYXJldHJhIGp1c3RvIHZlc3RpYnVsdW0gZGlhbS4=
            </openoces:Value>
          </ds:SignatureProperty>
          <ds:SignatureProperty>
            <openoces:Name>host</openoces:Name>
            <openoces:Value Encoding="base64" VisibleToSigner="yes">TmVtTG9nLWlu</openoces:Value>
          </ds:SignatureProperty>
          <ds:SignatureProperty>
            <openoces:Name>logonto</openoces:Name>
            <openoces:Value Encoding="base64" VisibleToSigner="yes">TmVtTG9nLWlu</openoces:Value>
          </ds:SignatureProperty>
          <ds:SignatureProperty>
            <openoces:Name>openoces_opensign_environment_browser_version</openoces:Name>
```

```
              <openoces:Value Encoding="base64" VisibleToSigner="no">MS4x</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_os_name</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">V2luZG93cyA3</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_java_vendor</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">
                T3JhY2xlIENvcnBvcmF0aW9u
              </openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_applet_context</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">
                Y29tLnN1bi5kZXBsb3kudWl0b29sa2l0LmltcGwuYXd0LkFXVEFwcGxldEFkYXB0ZXIkQXBwbG
                V0Q29udGV4dEltcGxAZmYxZTg1
              </openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_locale</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">ZW5fVVM=</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_applet_version</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">MS44LjU=</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_applet_digest</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">
                dG9kbzogaW1wbGVtZW50
              </openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_java_version</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">MS43LjBfMDc=</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_layout_signtext_fontsize</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">MTI=</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_layout_color_background</openoces:Name>
              <openoces:Value Encoding="base64"
VisibleToSigner="no">MjM4LDIzOCwyMzg=</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_browser_vendor</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">T3JhY2xl</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_layout_size_height</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">NDUw</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>action</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">c2lnbg==</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_layout_size_width</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">NTAw</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_local_time</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">
                TW9uIE9jdCAyMiAwODo0Nzo0NSBDRVNUIDIwMTI=
            </openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_environment_browser_name</openoces:Name>
              <openoces:Value Encoding="base64"
VisibleToSigner="no">c3VuLnBsdWdpbg==</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>openoces_opensign_layout_signtext_fontname</openoces:Name>
```

```
              <openoces:Value Encoding="base64"
VisibleToSigner="no">RGlhbG9nLnBsYWlu</openoces:Value>
            </ds:SignatureProperty>
            <ds:SignatureProperty>
              <openoces:Name>challenge</openoces:Name>
              <openoces:Value Encoding="base64" VisibleToSigner="no">
                NmUyMmIwODYtMzFmZC00MTgxLTg1ZGEtM2VjNDA3YmE1MzY5
              </openoces:Value>
            </ds:SignatureProperty>
          </ds:SignatureProperties>
        </ds:Object>
      </ds:Signature>
    </openoces:signature>
  </SigningAppletOutput>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
      <Reference URI="#SignedByKFOBS">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
        <DigestValue>SsMeozAA4WXcDpKN7ldVCToOq+UTjIirAot60J0/LDA=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
      bW+IPO8nnF8arOHe3alokimw58KBB7kW53S+Nn+wX0ye5xcC+x/m5bZuwrttkLOXKH7vPpUIotMOsULbfxlz
      XczliX+VA2LCeFfjuQOWpaJwcrLNp0NFoBoWWo4tqHvI0Jbh621sfoK5ULHIRTrYTGxOwJ6KU6hiHQoN269K
      4a4=
    </SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>
          MIIE5DCCBE2gAwIBAgIEQDghvTANBgkqhkiG9w0BAQUFADA/MQswCQYDVQQGEwJESzEMMAoGA1UEChMD
          VERDMSIwIAYDVQQDExlUREMgT0NFUyBTeXN0ZW10ZXN0IENBIElJMB4XDTEyMDUyMjEyNDEyOVoXDTE0
          MDUyMjEzMTEyOVowfDELMAkGA1UEBhMCREsxITAfBgNVBAoTGE5OSVQgQS9TIC8vIENWUjoyMTA5MzEw
          NjFKMCEGA1UEAxMaTk5JVCBBL1MgLSBLRk9CUyAtIFNpZ25pbmcwJQYDVQQFEx5DVlI6MjEwOTMxMDYt
          VUlEOjEzMzc2OTE1Njc0NjEwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKppY+YHJ8px8MlAP0U3
          CdXvxvhujG0nA3wgcSzoRC2YPtI1pRU0qsQepozq4Vz9K59FousOG/eMmgrii0FqfqtZh8Dv24mmr1HX
          cGYqbGgrIsgL6ebEq2zzCmICka3FYCoF11lbFSI6/7uvai4hPrSeTq5+5g3FjueFQOvuXrF1AgMBAAGj
          ggKuMIICqjAOBgNVHQ8BAf8EBAMCA7gwKwYDVR0QBCQwIoAPMjAxMjA1MjIxMjQxMjlagQ8yMDE0MDUy
          MjEzMTEyOVowRgYIKwYBBQUHAQEEOjA4MDYGCCsGAQUFBzABhipodHRwOi8vdGVzdC5vY3NwLmNlcnRp
          ZmlrYXQuZGsvb2NzcC9zdGF0dXMwggEDBgNVHSAEgfswgfgwgfUGCSkBAQEBAQEBAzCB5zAvBggrBgEF
          BQcCARYjaHR0cDovL3d3dy5jZXJ0aWZpa2F0LmRrL3JlcG9zaXRvcnkwgbMGCCsGAQUFBwICMIGmMAoW
          A1REQzADAgEBGoGXVERDIFRlc3QgQ2VydGlmaWthdGVyIGZyYSBkZW5uZSBDQSB1ZHN0ZWRlcyB1bmRl
          ciBPSUQgMS4xLjEuMS4xLjEuMS4xLjEuMy4gVERDIFRlc3QgQ2VydGlmaWNhdGVzIGZyb20gdGhpcyBD
          QSBhcmUgaXNzdWVkIHVuZGVyIE9JRCAxLjEuMS4xLjEuMS4xLjEuMS4zLjAcBglghkgBhvhCAQ0EDxYN
          b3JnYW5XZWJOb0RpcjCBlwYDVR0fBIGPMIGMMFegVaBTpFEwTzELMAkGA1UEBhMCREsxDDAKBgNVBAoT
          A1REQzEiMCAGA1UEAxMZVERDIE9DRVMgU3lzdGVtdGVzdCBDQSBJSTEOMAwGA1UEAxMFQ1JMMzIwMaAv
          oC2GK2h0dHA6Ly90ZXN0LmNybC5vY2VzLmNlcnRpZmlrYXQuZGsvb2Nlcy5jcmwwHwYDVR0jBBgwFoAU
          HJgJRxpMOLkQxQQpW/H0ToBqzH4wHQYDVR0OBBYEFKQTOvuQMJitEXACDYG9WwdiVvxMMAkGA1UdEwQC
          MAAwGQYJKoZIhvZ9B0EABAwwChsEVjcuMQMCA6gwDQYJKoZIhvcNAQEFBQADgYEAFocKONANXycyLjf+
          ECUQLI0RXlnv2nbcpOqIga8EusRDzmunap1dQcoE7zwvm2hCY87/Okqj6Yde/akQKqr6PGKiM4f/sROR
          Tx6U+rxgjVWPZWB2vPm/f+It8kj0v8Q/fuycAcRidm0owrCHPwgUQWVPMKnLrVbgDWqtbuo6Toc=
        </X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
</SignatureProof>
```

The signature proof (excluding the sign text) will be stored by the signing service for later retrieval together with a system proof. The signature proof together with the system proof is designed to provide irreputable evidence that the signing operation has occurred.

The SignedSignatureProof stored internally in the signing service will exclude the actual sign text because the service for privacy reasons is not allowed to persistently log the clear text signed by users. It is therefore the responsibility of the service provider to log this in his own system, such that in case of a dispute, the service provider can present the original sign text to be verified against the signature proof. The sign text can be acquired by using the following XPath expression:

```
/SignatureProof/kfobs:SigningAppletOutput/openoces:signature/ds:Signature/ds:Object/ds:Signatu
reProperties/ds:SignatureProperty[openoces:Name = "signtext"]
```

In order to be able to retrieve the signature and system proofs at a later point in time, each signature and system proof pair is an assigned two unique numbers called CA1 and CA2. The service provider must be able to provide these two numbers, since it will otherwise be impossible to retrieve the system and signature proofs.

CA1 value can be acquired using the following XPath expression:

```
/SignatureProof/@CA1
```

CA2 value can be acquired using the following xpath expression:

```
/SignatureProof/@CA2
```

Note: If the service provider fails to accurately store the base64 encoded sign text along with the CA1 and CA2 values in his own system, the data logged by the signing service has no proof value since the sign text is unknown!

Service providers are therefore strongly advised to test that the logged sign text in their own system match the digest in the received signature proof.

7 Checks required by Service Provider

In order to interact securely with the signing service, the service provider application must at minimum perform the following actions and checks:

1. Log expected SignText and generate a random RequestID value (e.g. a 128 bit pseudo random number) before redirecting the browser to the signing service.
2. Upon receiving the response from the signing service (via the browser):
    a. Validate that the response was signed by the specific VOCES certificate published for the signing service (not just any VOCES certificate).
    b. Check that the signing service certificate has not been revoked and has not expired.
    c. Check that the Status is "OK".
    d. Check that the RequestID in the response matches the expected RequestID for the given user, and then mark that the given RequestID cannot be received again.
    e. Validate that the sign text was actually signed by the user (the corresponding value of the <openoces:Name>signtext</openoces:Name>) is the same as the expected sign text logged in step 1 (comparison of base64 encoded strings).
    f. As a minimum log CA1, CA2 and sign text, or preferably the entire <SignatureProof> element.

    Besides checks performed in the application, the service provider must also protect their systems and infrastructure properly. This includes for example the private key for the VOCES or FOCES certificate used to authenticate the service provider to the NemLog-in signing service.

8 Error Codes

Error code	Description
REVOKED	The user signed with a certificate that has been revoked.
WRONG_SERIAL_NUMBER	The user signed with a certificate which had a different serial number than the one specified by the service provider.
UNSUPPORTED_CERTIFICATE	The user did not sign with a valid OCES certificate.
EXPIRED	The user used a certificate that has expired.
NOT_YET_VALID	The user used a certificate that is not yet valid.
VALIDATION_ERROR	The service provider certificate provisioned into signing service (referenced by entity Id) is not valid.
INVALID_CERTIFICATE	The user used an invalid certificate.
INVALID_PARAMETER	A mandatory parameter is missing or an invalid parameter was specified by the service provider.
APPLET_FAILURE	The Nets DanId signing applet failed.
INVALID_FINGERPRINT	The signed fingerprint specified by the service provider was invalid.
TEXT_MISMATCH	Unable to validate the signature against the sign text.
SERVICEPROVIDER_NOTREGISTERED	The entity id specified by the service provider was not known (i.e. has not been registered through the Connection Support System).
UNSUPPORTED_DIGEST_ALGORITHM	The digest algorithm specified by the service provider is not valid. Must be either SHA-1, SHA-256 or unspecified.
INTERNAL_ERROR	An error occurred that prevented the signing service from creating or storing the signature and system proofs.
SERVICE_DENIED	An error occurred that prevented the signing service from creating or storing the signature and system proofs.

9 Signing Web Service

The signing web service allows the service provider to send signature proofs to the signing service, in case the service provider has created this proof himself, e.g. by providing a rich user interface for signing agreements. The signing web service uses the same signature verification component as the signing web page, but the verification process is weakened because part of the process (the use of the signing applet) is not under the control of the signing web service.

The signing web service returns a signed signature proof as described in section 5 above. The service provider must save the sign text, CA1 and CA2 values of the signature proof to be used in case of a dispute.

The WSDL description for the signing web service can be obtained from the following link (the endpoint referred point to the integration test environment):

https://signingservice.signering.test-nemlog-in.dk/SigningService.svc?wsdl

The web service exposes a single operation called *ValidateSignature*.

ValidateSignature has the following input parameters:

Field name	Description	Mandatory	Processing
Signature	The signature text as it was received from the DanID applet Please note that it is important to keep white space **exactly** as it was received from DanID, as the validation will otherwise fail.	Yes	Signature = Base64 encoded signed document received from DanID.
EntityId	Each service provider has a unique EntityId which is used to identify the service provider in the federation	Yes	
SignedFingerPrint	A signed digest of the fields EntityId and Signature. The digest must be signed using the service provider's private key.	Yes	SignedFingerPrint = Base64Encode(SignSha256WithRsa(Sha256(Utf8Encode(Concat(EntityId, Signature))))

The output of the *ValidateSignature* operation is a structure containing the following fields:

Field name	Description	Mandatory	Processing
StatusCode	Indicates the result of the signing operation: OK: The user signed the sign text. CANCELLED: The user pressed the cancel button. ErrorCode: *An error occurred. The error code indicates the type of error.*	Yes	Error codes are described in section 8.
PID	In case a citizen certificate was used for signing, this field contains the PID of the certificate used	No	
CVR	In case an employee certificate was used for signing, this field contains the CVR number of the certificate used	No	
RID	In case an employee certificate was used for signing, this field contains the RID of the certificate used	No	
SignedSignatureProof	The signed document received by DanId and re-signed by the signing service.	No	Encoded according to: Base64Encode(Utf8Encode(SignedSignatureProof))
SignedFingerPrint	A signed digest of the fields StatusCode, EntityId, PID, CVR, RID, SignedSignatureProof. The digest is signed with the signing service's certificate.	Yes	SignedFingerPrint = Base64Encode(SignSha256WithRsa(Sha256(Utf8Encode(Concat(StatusCode, EntityId, PID, CVR, RID, Base64Encode(Utf8Encode(SignedSignatureProof)))))))