STORK 2.0 Signature overview

Confidentiality note:

Excerpt of STORK project-internal information. Do not share until the underlying specification are stable and get published.

Table of contents

List of abbreviations

<Abbreviation>	<Explanation>
AP	Attribute Provider
eID	Electronic Identity
EU	European Union
MS	STORK2.0 Member State
MW	MiddleWare
PEPPOL	Pan-European Public Procurement Online
PEPS	Pan European Proxy Server
SAML	Security Assertion Markup Language
SP	Service Provider
SPOCS	Simple Procedures Online for Cross- Border Services
STORK 2.0	Secure idenTity acrOss boRders linKed 2.0
V-IDP	Virtual Identity Provider
QAA	Attribute Quality Authentication Assurance

1 Introduction

STORK and STORK2.0 are EU-projects which aim to…

> *…establish a European eID Interoperability Platform that will allow citizens to establish new e-relations across borders, just by presenting their national eID.*

The purpose of this document is to provide a high-level overview of the STORK2 architecture to provide a better understanding of the signature related interfaces and technologies. This document further goes into technical details on signature creation related parts.

1.1 STORK2 Overview

A (simplified) overview of the components involved in the STORK infrastructure is shown in Figure 1. The figure shows three different roles:

- The Service Provider (SP) providing the application.
- The SP interfaces either to a national central Pan-European-Proxy-Service (S-PEPS) or a decentralized Virtual Identity Provider (V-IDP for middleware countries like Austria and Germany).
- The Pan-European-Proxy-Service in the citizen's home country (C-PEPS) that provides the eID means.

In the simplest case the service provider needs to identify the citizen, regardless of its home country and their national eID solution. Therefore the SP sends an authentication request to the S-PEPS which is either handled locally or sent to the C-PEPS of the citizen. This is a completely transparent process for the SP. For signature-creation, the same architecture is used.

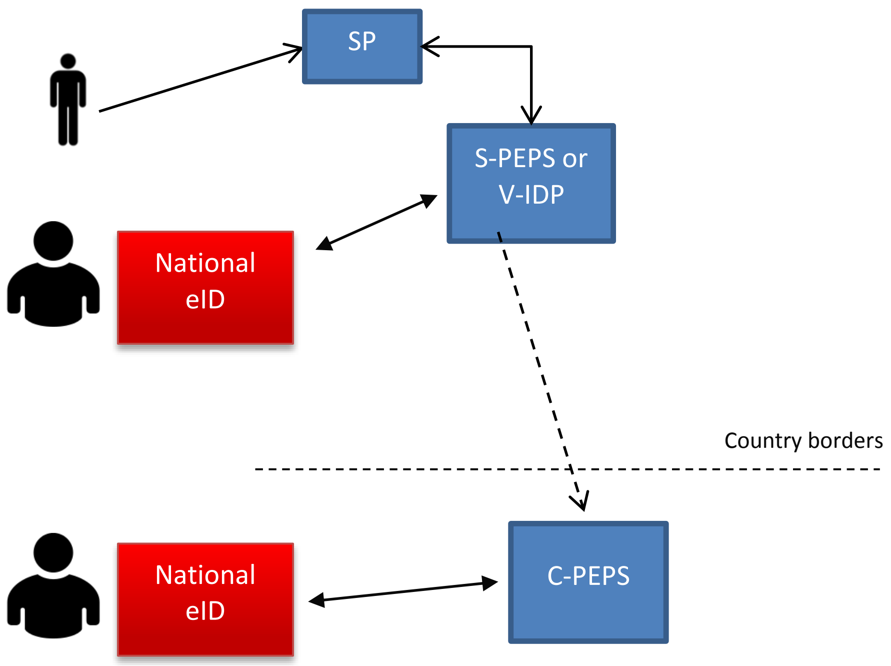


Figure 1: Simplified STORK 2.0 overview

The functionality of STORK2.0 is basically defined by the following high level business cases:

1. **Authentication on behalf of** is the process that allows a user to access privileged data of the represented person. Usually this process ends with a fully identified user (representative) and represented person, which means that their eID data is transferred to the service provider (SP), and this SP recognizes this user as a representative of a known customer, student, partner, or whatever relationship this represented person may have with the SP.

   Just like in STORK1, the service provider may determine the data he needs in this authentication, especially as mandates are so often “nearly absolute powers”. E.g. a foreign hospital may require someone to have a mandate to act on behalf of a medical institution in his own country, and also to be a medical doctor (and not being e.g. a general manager of financial manager). Thus this process is the same as *registration on behalf of*.

2. **Powers (for digital signature)** is the process that allows a service provider to verify that a user has enough power to represent the represented person, in case he has received the digital signature of the representing person on behalf of the represented person. This process is very similar to previous process; the main difference is the initiating action: in this case it’s the reception by SP of a digital signature.

3. **Business attributes** is a review of the authentication process of STORK1, but including “business attributes”, i.e. attributes which don’t come from a one and only central authority; instead they may come from universities, hospitals, etc.

Signature creation and verification is one key component of STORK 2.0 and is illustrated in chapter 2 and chapter 3.

2 Signature Creation

The STORK2-Interface provides a Create-Signature-method for signing arbitrary data using a citizens' certificate-based eID token[1] (or signature-creation device in general). Thanks to this method a SP (through S-PEPS or V-IDP) is able to call this method finally provided by the citizen's C-PEPS or MW.

Three cases are supported, which support different use cases:

1. Signing data during authentication: This has already been supported in STORK 1, e.g. to sign a proof of receipt in eDelivery. STORK 2.0 extends this technically by further document formats and introducing signature qualities the SP may need.
2. Signing data after authentication: This use case allows signed transactions during an authenticated session. Examples are money transfers in an authenticated Internet banking session, signing a tax declaration after having logged in to the tax portal, or signing a form at an eGovernment portal. These cases are extensions to STORK1 now supported by STORK 2.0.
3. Signing data independent of prior authentication: The same functions can be used independent of the authentication, such as signing a contract or an ad-hoc mandate right before using it in an STORK 2.0 "on behalf" authentication. This also is an extension made by STORK 2.0. It is introduced to allow and support additional use cases without major additional efforts, as the underlying functionalities anyhow are implemented for the two cases above.

In order to maintain backward compatibility, the STORK1 sign-on-authentication request is used for case 1. above. For the additional two use cases 2. and 3., an OASIS-DSS interface is used. OASIS-DSS has already been introduced in STORK1 for case 1. above, it is now further profiled to support case 2. and 3.

In addition, the STORK 2.0 signature creation function gives SPs more control over the signature request in order to adapt to its business process needs. In a nutshell – further detailed in this technical specification – the SPs enhanced control includes:

- The SP can request the signature quality, as needed for its business process. We distinguish three qualities: (1) qualified electronic signatures (QES); (2) Advanced Electronic Signatures based on qualified certificates (AdES+QC); and (3) Advanced Electronic Signatures (AdES). The three cases are distinguished via the certificate quality: A qualified certificate policy (QCP) supporting qualified signatures (referred to "OCP public + SSCD" or "QCP+" or equivalent) supports QES, a QCP public or equivalent supports AdES+QC, and non-qualified certificate policies support AdES.
- The SP can request the document and signature formats. The specification defines raw text, XML forms and PDF with the signature formats CAdES, PAdES, and XAdES. The envisaged STORK 2.0 implementation however limits itself to PAdES for arbitrary documents (including figures, etc.) and XAdES for raw text and simple forms.
- A "strong binding" option is specified to allow SPs that maintain high-value or critical processes to ensure that the same person that authenticated to a session actually signed at the end of the process (2. Signing data after authentication above). This gives SP better control to meet session hijacking or substitution attacks in mission-critical scenarios. The strong binding option uses the fact that most qualified certificates in STORK either hold the identifier used to create the "storkID", or a reference to it is given.

[1] Not all STORK 2.0 eID are signature-creation devices, though the majority is. The consortium is aware that not all STORK eID can use this function. It still is not discriminatory to support it, as if a SP's has a written-form and thus signature requirement, electronic processing without a signature is ruled out anyhow or needs a manual backup process to hand in signed paper.

The option and suggestions on when and how SPs can use it are given in this specification.

2.1 Signature Creation Workflows

STORK1 already has the capability to sign documents. This section describes the current state and the extensions to the STORK1 approach regarding the signature creation workflows.

The signature request follows the OASIS-DSS specification, but is not limited to a restricted profile (for more information regarding OASIS-DSS refer to section 2.1.1 and 2.3). The OASIS-DSS request <dss:SignRequest> is wrapped inside a <stork:RequestedAttribute> Element. The following listing shows an example attribute request.

```
...
<stork:RequestedAttribute
Name="http://www.stork.gov.eu/1.0/signedDoc"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
isRequired="false"
 <saml:AttributeValue>
 <dss:SignRequest>...</dss:SignRequest>
 </saml:AttributeValue>
</stork:RequestedAttribute>
...
```

2.1.1 Signature Creation on Authentication

In this case, the signature creation process takes place during the SAML authentication request. The OASIS-DSS signature request is embedded in a STORK2 <stork:RequestedAttribute> Element.

2.1.1.1 Example Request

This example request only shows the signature creation relevant parts.

```
<saml2p:AuthnRequest
  xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:stork="urn:eu:stork:names:tc:STORK:1.0:assertion"
  xmlns:storkp="urn:eu:stork:names:tc:STORK:1.0:protocol"
  AssertionConsumerServiceURL="https://S-
PEPS.gov.xx/PEPS/ColleagueResponse"
  Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified"
  Destination="http://C-PEPS.gov.xx/PEPS/ColleagueRequest"
  ForceAuthn="true"
  ID="390205d2-ea52-4aaa-966c-61f312131ddc"
  IsPassive="false"
  IssueInstant="2010-02-03T17:06:18.521Z"
  ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
  ProviderName="University Oxford"
  Version="2.0">
...

   <saml2p:Extensions
xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol">
...
      <storkp:RequestedAttributes>
         <stork:RequestedAttribute
      Name="http://www.stork.gov.eu/1.0/signedDoc"
      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
      isRequired="false"
```

```
        <saml:AttributeValue>
        <dss:SignRequest>...</dss:SignRequest>
        </saml:AttributeValue>
       </stork:RequestedAttribute>
...
       </storkp:RequestedAttributes>
    </stork:Extensions>
</saml2p:AuthnRequest>
```

2.1.2 Signature Creation with optional Authentication

The former described signature creation method (section 2.1.1) requires the SP to issue a SAML authentication request to invoke the signature creation workflows. For business cases where digital signatures are not required during the authentication phase or no authenticated session is required at all, this method is not practicable. Hence, a further signature creation workflow is specified (tightly aligned with the former workflow), which directly uses an HTTP POST enabled OASIS-DSS interface without embedding the request in the SAML request. For a detailed specification on the OASIS-DSS profile refer to section 2.2 and 2.1.1.

2.1.2.1 OASIS-DSS HTTP POST Transport Binding

To support both a signature request during an authenticated session and as part of non-authenticated sessions, the OASIS "HTTP POST Transport Binding" as specified in section 6.1 of **Error! Reference source not found.** is used.

The "TLS Security Binding" with "TLS X.509 Server Authentication" MUST be used (section 6.3.1 of [3])

This transport binding has been chosen, as OASIS-DSS HTTP POST matches with the SAML HTTP POST binding already used by STORK, but it also supports issuing the signature-creation request from non-authenticated sessions.

2.2 STORK 2.0 OASIS DSS Profiles for XAdES and CAdES

The signature formats to be supported are XAdES, CAdES and PAdES, in the following variants: PAdES-3, XAdES BES/EPS, CAdES BES/EPS.

Whereas the definitions of XAdES and CAdES support in the OASIS DSS are already available as a part of OASIS DSS AdES Profile [3], the new extension profile for PAdES is additionally defined. Furthermore, we introduce the additional elements necessary to support the functionality and use cases foreseen by this document.

AdES (Advanced Electronic Signature) abstract profile is based on OASIS DSS Core Profile. The base form of the profile is further refined in XML Advanced Electronic Signatures concrete profile and CMS-based Advanced Electronic Signature profile. They both support creation and verification of advanced signatures as defined in XAdES (ETSI TS 101 903) and CAdES (ETSI TS 101 733), including the update of advanced signatures by addition of unsigned properties.

2.2.1 XML Advanced Electronic Signatures concrete Profile

The element SignRequest is sent by the client to request a signature on input documents.

Table 1 contains the refined list of attributes and the elements defined in this request by OASIS DSS necessary to support STORK use cases in case of XAdES signatures. The definitions listed here refer to required elements only, whereas some of them have been marked as optional in original OASIS DSS specification. The optional elements, as defined in OASIS DSS and its XAdES profile, should stay optional e.g. recognized by the service but not critical for the functionality of the service in any case.

Element	Allowed attribute set	Remarks
Attribute *RequestId*	*REQUIRED*	RequestId SHOULD be globally unique.
Attribute *Profile*	*REQUIRED*	Indicates a particular profile used for signature request `(urn:oasis:names:tc:dss:1.0:profiles:XAdES)`
Element OptionalInputs	REQUIRED	Used to include additional information supporting STORK use cases
Element AdditionalProfile	*REQUIRED*	Defines additional profile containing extended STORK 2.0 definitions. The STORK 2.0 profile here to be used is identified with `urn:stork20:dss:XAdES`, which should be denoted as a value of this field
Element Language	*REQUIRED*	Describes language used for further interaction with the client in the terms of localization
Element SignatureType	*REQUIRED*	For XML signatures the value of this element must be equal to `urn:ietf:rfc:3275`
Element InputDocuments	*REQUIRED*	This element contains one or multiple input documents which are to be sent to a MS signature service

Table 1: OASIS-DSS XAdES profile for STORK

2.2.2 CMS-based Advanced Electronic Signature profile

This subsection deals with the definition of the CAdES SignRequest for the particular case of CAdES signatures.

Table 2 contains the refined list of attributes and the elements defined in this request by OASIS DSS necessary to support STORK use cases in case of CAdES signatures. The definitions listed here refer to required elements only, whereas some of them have been marked as optional in original OASIS DSS specification. The optional elements, as defined in the OASIS DSS and its CAdES profile, should stay optional e.g. recognized by the service but not critical for the functionality of the service in any case.

Element	Allowed attribute set	Remarks
Attribute *RequestId*	*REQUIRED*	RequestId SHOULD be globally unique.
Attribute *Profile*	*REQUIRED*	Indicates a particular profile used for signature request `(urn:oasis:names:tc:dss:1.0:profiles:CAdES)`
Element OptionalInputs	REQUIRED	Used to include additional information supporting Stork use cases
Element AdditionalProfile	*REQUIRED*	Defines additional profile containing extended Stork definitions. The Stork profile here to be used is identified with `urn:stork20:dss:CAdES` , which should be denoted as a value of this field

Element Language	*REQUIRED*	Describes language used for further interaction with the client in the terms of localization
Element SignatureType	*REQUIRED*	For CMS signatures the value of this element must be equal to `urn:ietf:rfc:3369`
Element InputDocuments	*REQUIRED*	This element contains one or multiple input documents which are to be sent to a MS signature service

Table 2: OASIS-DSS CAdES profile for STORK

2.3 STORK 2.0 OASIS DSS Profile for PAdES

The support for PAdES is not originally defined as a part of OASIS DSS Advanced Electronic Signature Profiles. Therefore, we extend the OASIS DSS Advanced Electronic Signature abstract profile and on its basis define the PDF Advanced Electronic Signatures concrete profile.

Element	**Allowed attribute set**	**Remarks**
Attribute *RequestId*	*REQUIRED*	RequestId SHOULD be globally unique.
Attribute *Profile*	*REQUIRED*	Indicates a particular profile used for signature request `urn:oasis:names:tc:dss:1.0:profiles:CAdES PAdES`
Element OptionalInputs	REQUIRED	Used to include additional information supporting STORK 2.0 use cases
Element AdditionalProfile	*REQUIRED*	Defines additional profile containing extended Stork definitions. The Stork profile here to be used is identified with `urn:stork20:dss:PAdES`, which should be denoted as a value of this field
Element Language	*REQUIRED*	Describes language used for further interaction with the client in the terms of localization
Element SignatureType	*REQUIRED*	In the cases of PAdES 3-4 signatures the value of this element must be equal to `urn:ietf:rfc:3369`
Element InputDocuments	*REQUIRED*	This element contains one or multiple input documents which are to be sent to a MS signature service

Table 3: STORK PAdES profile based on OASIS-DSS

2.3.1 STORK 2.0 OASIS-DSS extensions

STORK 2.0 amends the OASIS-DSS request mainly in two aspects:

1. A “strong identity binding” in an authenticated session (StrongBindToIdentity)
2. The signature quality is derived from the PEPPOL signature policies[2]. The scope is limited to the 3 cases:
    a. Qualified certificates based on a SSCD (QCP+ for QES), this corresponds to PEPPOL Certificate Quality level 6
    b. Qualified certificate (QCP for AdES based on qualified certificates), this corresponds to PEPPOL Certificate Quality level 5

c. Non-qualified certificate, this corresponds to PEPPOL level 3

Contrary [2] the parameters *IndependentAssurance*, *HashAlgQuality*, and *PublicKeyAlgQuality* are NOT USED. This as for qualified certificates signature suites are determined by the certification service provider, its supervision is determined by the MS. This limits choices by the SP and can hinder interoperability.

The elements defined in Table 4 represent an extension defined to support STORK 2.0 use cases and requirements. These elements are used in conjunction with previously described XAdES/CAdES/PAdES profiles, included in the same SignRequest.

Element	Allowed attribute set	Remarks
Attribute *DocumentURL*	*OPTIONAL*	References the location of the document to be signed, stored on SP locally
Attribute *ReturnURL*	*OPTIONAL*	Specifies the URL the document is put after the signature creation process completed
Attribute StrongBindToIdentity	*OPTIONAL*	The value of this attribute contains the StorkID of the user; it is later.
Attribute EnforceIdentityBinding	*OPTIONAL*	The value of this boolean field determines, if the StrongBindToIdentity selection should be enforced.
Element QualityLevelRequirements	*OPTIONAL*	Signature quality
Element CertificateQuality	*REQUIRED*	This element contains the minimum required certificate quality level. STORK 2.0 aims at 3 levels: (1) QES, (2) AdES based on qualified certificates, and (3) AdES. This maps to the following CertificateQuality Levels in the PEPPOL profile [2] **6** very high: QCP+, QES (i.e. qualified cert. and SSCD) **5** very high: QCP **3** high: NCP or similar Other values (1, 2, 4) are NOT USED. **Note:** This specification sets as an additional requirement to the PEPPOL signature policy [2] for certificate quality: • Value 6 MUST only be used with an SSCD • Value 5 and 6 MUST be a qualified certificate. i.e. there are no “or similar” options.
Element IndependentAssurance	*NOT USED*	The signature quality requested by the SP is determined by the certificate quality.
Element HashAlgQuality	*NOT USED*	The signature quality requested by the SP is determined by the certificate quality.
Element PublicKeyAlgQuality	*NOT USED*	The signature quality requested by the SP is determined by the certificate quality.

Table 4: STORK-extensions to OASIS-DSS Request

The following Element is added as a valid InputDocument Element

Element	Allowed attribute set	Remarks
Element DocumentURL	*REQUIRED*	This element references the Document URL

Table 5: DocumentURL added as valid InputDocument

The following Table 6 represents the STORK 2.0 extensions used in SignResponse element.

Element	Allowed attribute set	Remarks
Attribute *DocumentWithSignature*	*OPTIONAL*	References the location where the signed document resides
Attribute IdentityBindSuccess	*OPTIONAL*	The value of this element can be one of the following: OK, FAILED, and INCOMPLETE. The value “OK” means that the signature service ensures that the signature has been created by a signature-creation device belonging to the person holding storkID used in the signature-request. “FAILED” means that another person than identified by storkID signed. INCOMPLETE indicates that the signing service cannot determine, as e.g. the signature-creation device is not bound to an ID that a storkID can be derived from. If the request used EnforceIdentityBinding, the infrastructure MUST NOT return a signed document if strong identity binding fails, i.e. if it results in FAILED or INCOMPLETE. *Note 1*: It is up to SP to determine, if the cases FAILED or INCOMPLETE allow for further automated processing, involve a manual process, or lead to terminating the business process. *Note 2*: The STORK infrastructure transports signed documents. No guarantees are given, that the signature is valid, i.e. no signature verification on the signed document has been performed by the STORK infrastructure.

Table 6: STORK-extensions to OASIS-DSS Response

2.4 Complete Signature Creation Workflow using OASIS-DSS Interface

Figure 2 shows the signature creation sequence. The SP, S-PEPS, C-PEPS and V-IDP directly communicate with each other in blocking service calls. Additionally the user gets from SP to S-PEPS, C-PEPS, V-IDP and finally to the signing app in a different session/thread. This approach has the advantage that no HTTP-POST redirections are required and that the OASIS-DSS interface can synchronously return the OASIS-DSS response as it is defined in the OASIS-DSS HTTP binding specification.

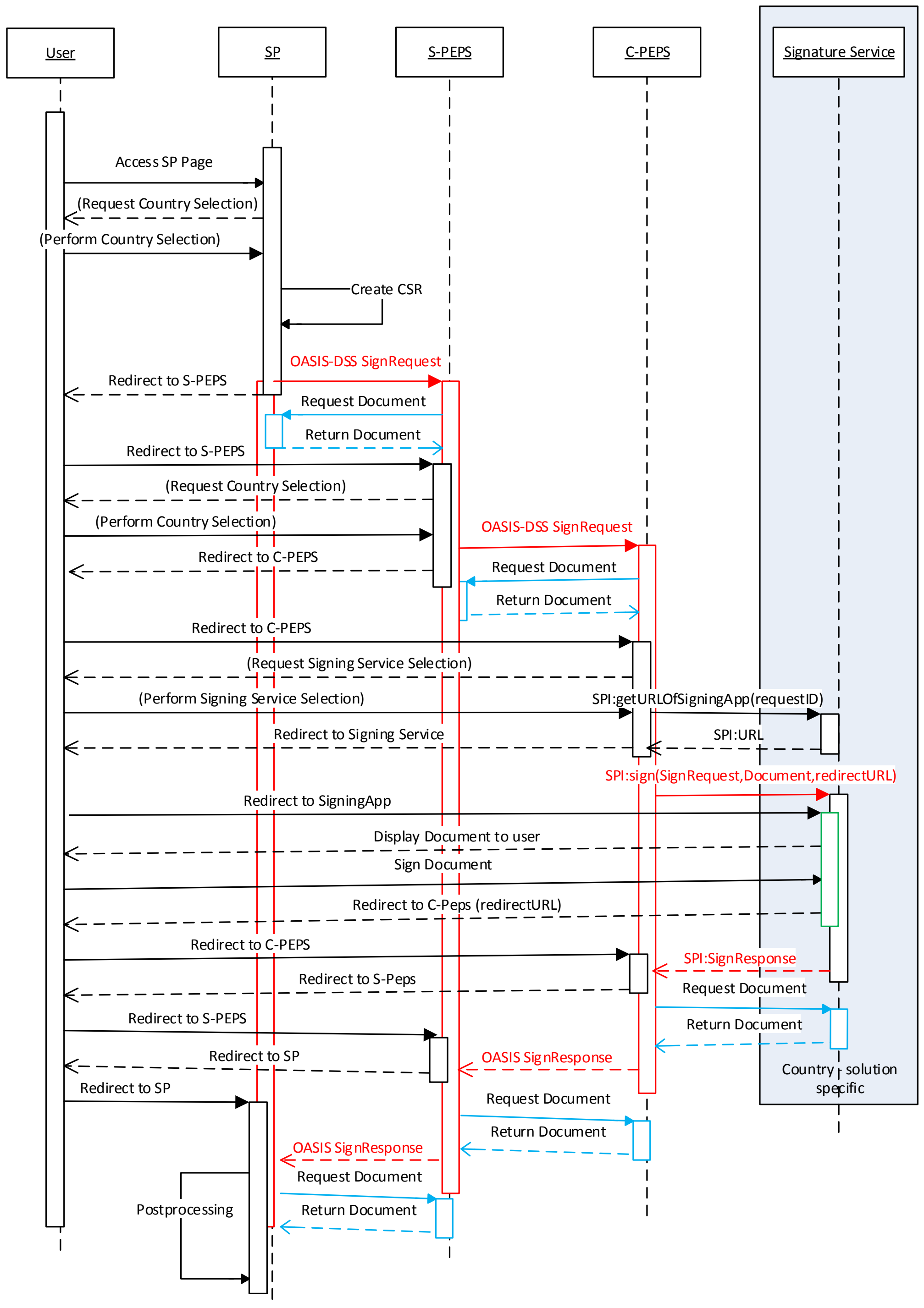
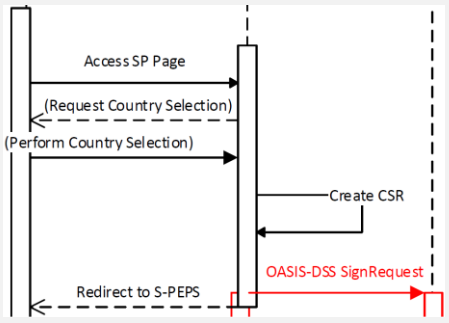
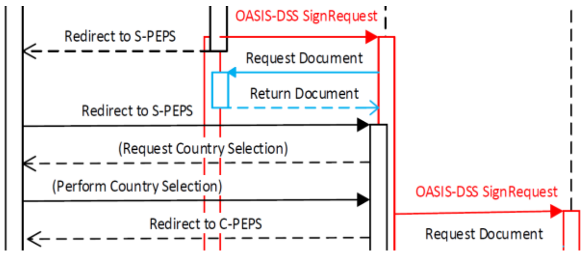
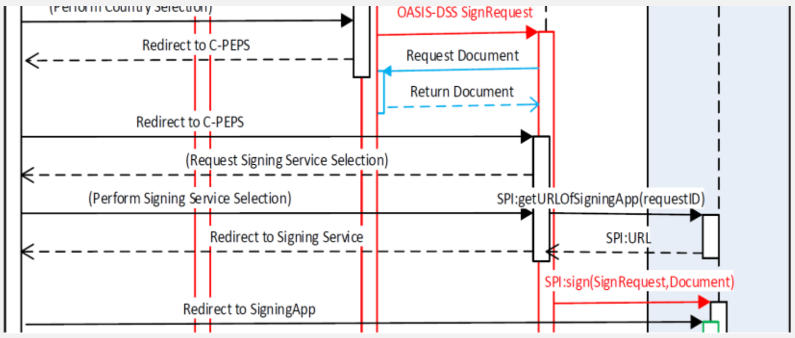
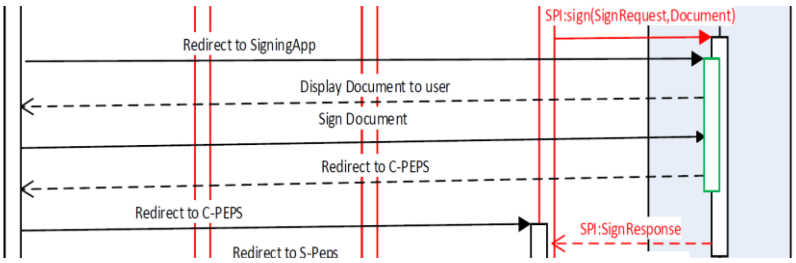
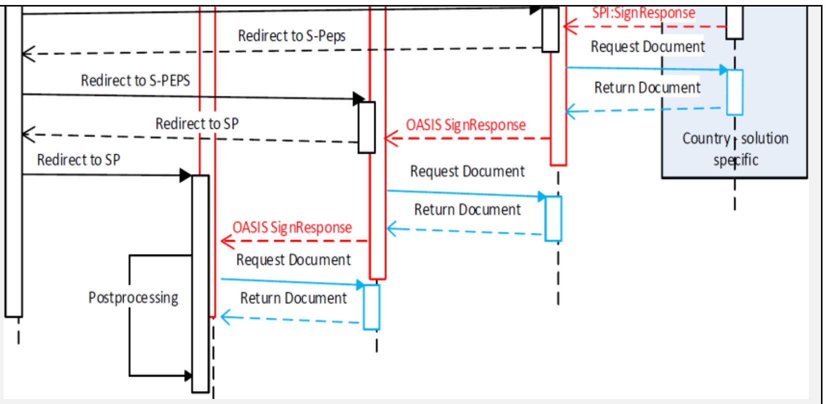


Figure 2: Signature Creation Sequence

1	Create and send SignRequest	**Description** The user uses a service offered by the service provider. The service provider optionally offers a country selection and sets the *TargetCountry* to the selected country. The service provider creates an OASIS-DSS *SignRequest* and sets the *ReturnURL*. The *ReturnURL* specifies to which URL the user is redirected after the signing process. If the document to be signed is larger than the limit the *DocumentURL* is set. After that the *SignRequest* is sent to the S-PEPS/V-IDP and the citizen browser is redirected to the S-PEPS/V-IDP. The redirect URL must contain the *RequestId* of the *SignRequest*.
		External Actors • The citizen browser is redirected to the S-PEPS/V-IDP • The S-PEPS/V-IDP receives the OASIS-DSS SignRequest
		Sequence Diagram  Access SP Page (Request Country Selection) (Perform Country Selection) Create CSR OASIS-DSS SignRequest Redirect to S-PEPS
2	Forward to C-PEPS/ V-IDP	**Description** The S-PEPS/V-IDP receives the *SignRequest*. It stores a mapping *RequestId->ReturnURL* and changes the *ReturnURL* to a newly generated one. If the *DocumentURL* is set, the S-PEPS/V-IDP loads the document and adapts the *DocumentURL*. The user is redirected to the S-PEPS/V-IDP. Because of the *RequestId* the S-PEPS/V-IDP knows which *SignRequest* belongs to this user. If the *SignRequest* contains no *TargetCountry* and the attribute *StrongBindToIdentity* is not set the S-PEPS/V-IDP requests a country selection form the user and stores the country code in the *TargetCountry attribute*. After that the *SignRequest* is sent to the C-PEPS/V-IDP of the corresponding country and the user is redirected to the C-PEPS/V-IDP. The redirect URL must contain the *RequestId* of the *SignRequest*.
		External Actors • The citizen browser is redirected to the C-PEPS/V-IDP • The S-PEPS/V-IDP receives the OASIS-DSS SignRequest
		Sequence Diagram  OASIS-DSS SignRequest Redirect to S-PEPS Request Document Return Document Redirect to S-PEPS (Request Country Selection) (Perform Country Selection) OASIS-DSS SignRequest Redirect to C-PEPS Request Document
3	Forward to Signing-App	**Description** The C-PEPS/V-IDP receives the *SignRequest*. It stores a mapping *RequestId->ReturnURL* and changes the *ReturnURL* to a newly generated one. If the *DocumentURL* is set, the C-PEPS/V-IDP loads the document. *The C-PEPS/V-IDP* checks whether the document meets the requirements. If the document does not meet the requirements the signing process is cancelled. The user is redirected to the C-PEPS/V-IDP. Because of the *RequestId* the C-PEPS/V-IDP knows which *SignRequest* belongs to this user. If the C-PEPS/V-IDP has more than one Signing Service it can optionally request a Signing Service selection. Using the SPI method getID() the C-PEPS/V-IDP can distinguish the different provider implementations. Then the C-PEPS/V-IDP calls the SPI method *getURLOfSigningApp(String requestID)* to receives the URL, to which the user should be redirected. Finally the C-PEPS/V-IDP calls the SPI method *sign* and passes the

		SignRequest and the document.
		External Actors • The citizen browser is redirected to the Signing Service • The Signing Service gets the *SignRequest* and the Document
		Sequence Diagram  (Perform Country Selection) OASIS-DSS SignRequest Redirect to C-PEPS Request Document Return Document Redirect to C-PEPS (Request Signing Service Selection) (Perform Signing Service Selection) SPI:getURLOfSigningApp(requestID) Redirect to Signing Service SPI:URL SPI:sign(SignRequest,Document) Redirect to SigningApp
4	Document Signing	**Description** The Signing Service gets the *SignRequest and the document*. The user is redirected to the Signing Service. The URL was previously generated by the Signing Service, therefore it knows to which *SignRequest* it belongs. The Signing Service displays the document to the user. The user signs the document and the Signing Service receives the signed document. The user is redirected to the C-PEPS/V-IDP using the *ReturnURL* specified in the *SignRequest*. This URL must contain the *requestID.* Finally the SPI method *sign* returns the *SignResponse* to the C-PEPS/V-IDP. If the document is larger than the limit the attribute *DocumentWithSignature* is set.
		External Actors • The user signs the document • The C-PEPS/V-IDP calls the SPI method sign
		Sequence Diagram  SPI:sign(SignRequest,Document) Redirect to SigningApp Display Document to user Sign Document Redirect to C-PEPS Redirect to C-PEPS SPI:SignResponse Redirect to S-Peps
5	Redirecting the user to the SP / Sending the SignResponse to the SP	**Description** The C-PEPS/V-IDP gets die *SignResponse*. If the attribute *DocumentWithSignature* is set the C-PEPS/V-IDP loads the document and changes the *DocumentWithSignature attribute.* Then the C-PEPS/V-IDP returns the *SignResponse* as response to the *SignRequest* (same session). The S-PEPS/V-IDP gets the *SignResponse* and sends it as response in the open session to the SP (same procedure as C-PEPS/V-IDP to S-PEPS/V-IDP). Than the SP receives the *SignResponse*. If necessary the SP loads the document from the S-PEPS/V-IDP. Concurrently the C-PEPS/V-IDP redirects the user to the S-PEPS/V-IDP. The C-PEPS/V-IDP gets the corresponding S-PEPS/V-IDP address via the *requestID* received via the user. The S-PEPS/V-IDP repeats the procedure and redirects the user to the SP. Both the user and the SignResponse arrive at the SP. The SP may block the user if the *SignResponse* has not arrived yet.
		External Actors
		Sequence Diagram

		 SPI:SignResponse Request Document Redirect to S-Peps Return Document Redirect to S-PEPS Redirect to SP OASIS SignResponse Country solution specific Redirect to SP Request Document Return Document OASIS SignResponse Request Document Postprocessing Return Document

3 Signature Verification

The concept for signature validation is to create instances (services) which are aware of all certificate authorities through trust lists [1] and therefore is aware of the trust states (qualified, non-qualified) of the certificate authorities. As seen in Figure 1the validation service supports two use cases:

- Signature validation where the whole document is transmitted to the validation service. This has the disadvantage that it may result in a greater roundtrip time, depending on the document size and that confidential information may get exposed.

- Signature validation where only the relevant information is transmitted: document hash, signing certificate. Using this method a drawback is that some more or less complex logic (depending on the document format) is required on the signature validation requestor side. To overcome this, a gateway solution can be. It dispatches the document, calculates the hash and redirects the request to the final validation service. The response gets routed back to the requestor through the gateway.

To not load the SP with operating gateways (minimal-invasive for the SP), we assume that the signed documents are transmitted to the S-PEPS (V-IDP). The S-PEPS can depending on the MS situation either:

- invoke a separate validation service and transmit the whole signed document,
- invoke a separate validation service and transmit the hash-value / certificate,
- or implement a full-fledged validation service

The interfaces for signature verification also use an adapted OASIS-DSS profile as described in chapter 3.1.

3.1 OASIS-DSS Profile Specification

This section reproduces the OASIS-DSS verify profile specification as described in [2].

3.1.1 VerifyRequest Element

Element	Allowed attribute set	Remarks
Attribute *RequestId*	*REQUIRED*	RequestId SHOULD be globally unique
Attribute *Profile*	*REQUIRED*	
Element *OptionalInputs*		
Element *QualityLevelRequirements*		*REQUIRED*
In cases where there are no requirements regarding quality, the requested quality SHALL be set to 0.		
Child element *CertificateQuality*	*REQUIRED*	This element contains the minimum required certificate quality level. STORK 2.0 aims at 3 levels: (1) QES, (2) AdES based on qualified certificates, and (4) AdES. This maps to the following CertificateQuality Levels in the PEPPOL profile [2]: **6** very high+: QCP+, QES (i.e. qualified cert. and SSCD) **5** very high: QCP **3** high: NCP or similar Other values (1, 2, 4) are NOT USED. **Note:** This specification sets as an additional requirement to the PEPPOL signature policy [2] for certificate quality: • Value 6 MUST only be used with an SSCD • Value 5 and 6 MUST be a qualified certificate. i.e. there are no “or similar” options.
Child element *IndependentAssurance*	*NOT USED*	
Child element *HashAlgQuality*	*NOT USED*	
Child element *PublicKeyAlgQuality*	*NOT USED*	
Element *RespondWith*		*REQUIRED*
This is an enumeration type. The enumeration identifiers denoted with REQUIRED MUST be specified.		
Subject	*REQUIRED*	This is the distinguished name (DN) of the holder of the certificate (subject field in a certificate).
IssuerName	*REQUIRED*	This is the DN of the certificate issuer.
CertificateSerialNumber	*REQUIRED*	This is the serial number in the certificate.
KeyValue	*OPTIONAL*	This is the certificate holder’s public key.
HashAlgorithm	*OPTIONAL*	For a signature verification, this hash algorithm is extracted from the signature, while in a certificate validation this hash algorithm is found in the signature algorithm field in the certificate.
X509CertificateChain	*OPTIONAL*	This is the certificate chain for the certificate.
SKI	*OPTIONAL*	The SKI (Subject Key Identifier) is the SKI from the certificate. This is an extension in the certificate, and it provides a means of identifying certificates that contain a particular public key.

Element	Allowed attribute set	Remarks
Attribute *RequestId*	*REQUIRED*	RequestId SHOULD be globally unique
Attribute *Profile*	*REQUIRED*	
KeyUsage	*OPTIONAL*	This is the key usage from the certificate.
ExtendedKeyUsage	*OPTIONAL*	This is the extended key usage from the certificate.
BasicConstraints	*OPTIONAL*	This is the basic constraints for the certificate.
ValidFrom	*OPTIONAL*	The certificate is valid from this date.
ValidTo	*OPTIONAL*	The certificate is valid to this date.
SignHash	*OPTIONAL*	This is the decrypted hash of the signature.
ContentHash	*OPTIONAL*	This is the hash of the content sent in the request.
Content	*OPTIONAL*	This is the content sent in the request.
X509CRL	*OPTIONAL*	This is the CRL used to validate the end user’s certificate. If other methods than CRLs are used for validation of certificates, this value will be N/A. Note: The return of CRLs may reduce the response time of a request due to the size of the CRL.
OCSP	*OPTIONAL*	If OCSP is used in the validation of a certificate, the entire OCSP response is given here. If OCSP is not used, this value will be N/A.
CRLUrl	*OPTIONAL*	This is the URL from which the CRL was downloaded.
CRLNumber	*OPTIONAL*	This is the CRLNumber of the CRL used for validation of the certificate.
Timestamp	*OPTIONAL*	This is the timestamp of a signature. If a time stamp is not used, the value will be N/A.
Element *GatewayRequester*		*REQUIRED*
To support the use of a validation gateway forwarding requests to a validation authority (VA) service the element *GatewayRequester* has been defined. The element uses the *IdentityType* type which is based on the *RequesterIdentity* element. The *GatewayRequester* element is OPTIONAL, but it is REQUIRED when a validation gateway is being used to forward a request to a VA service.		
Child element *Name*	*REQUIRED*	This is the name of the validation gateway the verification request is sent through. This element is of the type saml:NameIdentifierType. This SHOULD be filled with the CN from the validation gateway’s client authentication certificate used in the two-way SSL connection.
Child element *SupportingInfo*	*OPTIONAL*	This element MAY include an URI to the requester or responder to support an asynchronous version of this protocol.
Element *UseVerificationTime*		*OPTIONAL*

Element	Allowed attribute set	Remarks
Attribute *RequestId*	*REQUIRED*	RequestId SHOULD be globally unique
Attribute *Profile*	*REQUIRED*	
The element *UseVerificationTime* is already defined in [3].		
Element *ReturnVerificationTimeInfo*		*REQUIRED*
The element *ReturnVerificationTimeInfo* is already defined in [3].		
Element *RequesterIdentity*		*REQUIRED*
The element *RequesterIdentity* is already defined in [3]. This element is made *REQUIRED* in this profile. The element is used to identify the requester of the verification.		
Child element *Name*	*REQUIRED*	This is the name of the requester who requested the verification. This element SHOULD be the same as the CN in the client authentication certificate used in the security bindings.
Child element *SupportingInfo*	*OPTIONAL*	This element may be used in an asynchronous communication between the client and the server. This element will then contain a respond address element.
Element *InputDocuments*		
This profile supports only one document in one request, but the document may have an unlimited number of signatures. The document is added using an appropriate child element of the *InputDocuments* element following the definitions in [3]. This element is *REQUIRED* in this profile. Instead of sending the entire document, a document hash and the signatures may be used.		
Element *SignatureObject*		
The *SignatureObject* element is only used in relation with the *DocumentHash* element in *InputDocuments*.		

Table 7: Signature VerifyRequest

3.1.2 VerifyResponse Element

Element	*Allowed attribute set*	Remarks
Attribute RequestId	*REQUIRED*	This attribute is REQUIRED in this profile. The RequestID must be the same as the RequestID in the request.
Attribute Profile	*REQUIRED*	
Element *Result*		
Child element *ResultMajor*	*REQUIRED*	The values defined for *ResultMajor* in the core specification are used as specified there.
Child element *ResultMinor*	*REQUIRED*	For *ResultMajor* code Success the following *ResultMinor* codes are defined: • *IncorrectSignature*: This code should be used when one of the signatures fails to verify and the *OverallAssertionStatus* attribute in the *ContentVerifyInfo* element is *NotTrusted*. • *ValidMultiSignatures*: This code should be used when the signature verification are valid for all signatures For *ResultMajor* code *InsufficientInformation* the following *ResultMinor* codes are defined: • *CrlNotAvailable*: This code is used if the CRL is unavailable during verification of any of the signatures resulting in an *OverallAssertionStatus* attribute in the *ContentVerifyElement* set to Indeterminate. • *OcspNotAvailable*: This code is used if the OCSP responder is unavailable during verification of any of the signatures resulting in an *OverallAssertionStatus* attribute in the *ContentVerifyElement* set to Indeterminate.
Element *OptionalOutputs*		
Element *ResponderIdentity*		*REQUIRED*
Child element *Name*	*REQUIRED*	This element SHOULD be populated with the CN of the service's signing certificate or the SSL Server authentication certificate used in the SSL connection.
Child element *SupportingInfo*	*OPTIONAL*	Not in use in this profile.
Element *QualityLevel*		*REQUIRED*
Child element *CertificateQuality*	*REQUIRED*	This is the actual quality of the certificate(s) used for signing. In addition to the PEPPOL signature policy [2], the values need to ensure that Value **6**: MUST ONLY be used for QES (qual. cert + SSCD) Values **5** or **6** MUST ONLY be used for qualified certificates
Child element *IndependentAssurance*	*OPTIONAL*	This is the independent assurance level of the certificate(s) used for signing.
Child element *HashAlgQuality*	*OPTIONAL*	This is the quality of the hash algorithm used.

Child element *PublicKeyAlgQuality*	*OPTIONAL*	This is the quality of the public key algorithm used.
Element *ResponseItem*		*REQUIRED*
Child element *IntValue*	*OPTIONAL*	The *IntValue* element is of type. This element may be used for any *RespondWithEnum* item that has an integer value as result.
Child element *Value*	*OPTIONAL*	The *Value* element is of type. This element is for future use, and it can have multiple values against one *RespondWithEnum* item.
Child element *Base64Value*	*OPTIONAL*	The *Base64Value* element is of type. This is used when the result of the *RespondWithEnum* item can be returned in base64 format. This value is for example used when the *ResponseItem* Id is “KeyValue”.
Child element *StringValue*	*OPTIONAL*	The *StringValue* element is of type string. This is used when the *ResponseItem* element is presented in string format.
Attribute *Id*	*OPTIONAL*	The *Id* attribute is of type *RespondWithEnum*, and it is an optional attribute. This attribute is used to identify the specified *ResponseItem*.
Element *ContentVerifyInfo* The *ContentVerifyInfo* element is used to return information about all the signatures and the belonging certificates used in the signed document. The element is *REQUIRED* in a response using this profile. The element is of type *ContentVerifyInfoType* and it consists of the following child elements and attribute:		*REQUIRED*
Child element *VerifyInfo*	*REQUIRED*	Refer to *QualityLevel* above.
Child element *Reason*	*OPTIONAL*	The *Reason* element is only used when the *OverallAssertionStatus* attribute is set to either *NotTrusted* or *Indeterminate*. The element will point to the signature(s) that failed verification, and this MAY be done using the *Id* attribute in the *VerifyInfo* element like this: <Reason> ID1, ID3 </Reason> - where the first and third signatures fails verification. ID1 and ID3 are retrieved from the *Id* attribute in the different *VerifyInfo* elements used in the specific verification transaction.

Attribute *OverallAssertionStatus*	*REQUIRED*	The *OverallAssertionStatus* attribute is of type *ContentStatusEnum*. This attribute provides the overall result of the signature verification of the signed document. The following listing shows the different status messages and their meaning: • *Trusted*: The overall result is trusted. All signature verifications succeeded. • *NotTrusted*: The overall result is not trusted. One or more of the signatures failed verification. • *Indeterminate*: The overall result is indeterminate. The verification process could not determine if this is a trusted or not trusted request The following rules apply regarding the usage of OverallAssertionStatus: • If any of the signatures have the attribute *AssertionStatus* set to Invalid, the *OverallAssertionStatus SHALL* be set to *NotTrusted* • If one or more *Indeterminate* signatures are present, and none *Invalid*, the *OverallAssertionStatus* attribute *SHALL* be set to *Indeterminate* • If one or more *InsufficientQuality* signatures are present, and none *Invalid* and *Indeterminate*, the *OverallAssertionStatus* attribute *SHALL* be set to *NotTrusted*. • If all signatures have the attribute *AssertionStatus* set to *Valid*, the *OverallAssertionStatus SHALL* be
Element *VerifyInfo*		*REQUIRED*
Each signature present on the verified document has its own *VerifyInfo* element		
Child element *QualityLevel*	*REQUIRED*	Refer to *QualityLevel* above.
Child element *ResponseItem*	*REQUIRED*	Refer to *ResponseItem* above.
Child element *FailureReason*	*OPTIONAL*	The *FailureReason* element will be used if the *AssertionStatus* attribute is either *Invalid*, *Indeterminate* or *InsufficientQuality*. The element *SHOULD* return a code explaining the reason for failure.
Attribute *Id*	*REQUIRED*	The *Id* attribute is used to identify the different signatures used on the signed document, hence it is the id of the signature.

Attribute *AssertionState*	*REQUIRED*	The *AssertionStatus* attribute gives the status of each verified signature. Following states are defined: • *Valid*: The signature is valid. • *Invalid*: The signature is invalid. The failure reason will be given in the *FailureReason* element • *Indeterminate:* It is not possible to determine the status of the signature due to for example no available CRL or OCSP responder. • *InsufficientQuality:* The signature is valid, but the quality of either the certificate used and/or the signature algorithms (hash and public key) are lower than the requested quality.
Element *VerificationTimeInfo*		*REQUIRED*
The element *VerificationTimeInfo* is already defined in [2].		
Child element *VerificationTime*	*REQUIRED*	Used as described in [3].
Child element *AdditionalTimeInfo*	*REQUIRED*	Refer to *ResponseItem* above. This element shall be populated with the time the response message was formed while the *VerificationTime* is the time of verification. New value for the *Type* attribute in *AdditionalTimeInfo*: urn:oasis:names:tc:dss:1.0:additionaltimeinfo:responseTimeInstant The *Ref* attribute is NOT USED in this profile.

Table 8: Signature VerifyResponse

4 References

[1] Commission Decision of 16 October 2009 setting out measures facilitating the use of procedures by electronic means through the 'points of single contact' under Directive 2006/123/EC of the European Parliament and of the Council on services in the internal market, 2009/767/EC.

[2] http://project.peppol.eu/about_peppol, retrieved on 13/06/2013

[3] 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.html

[4] Policy requirements for certification authorities issuing qualified certificates, ETSI TS 101 456.