CGI Signature Service Documentation

Integration Guide 2011.8 Generated: 2021-02-22

Version 2011.8

Table of Contents

1. Introduction

1.1. Overview

This chapter gives an overview of CGI Signature Service and all the components that are involved.

1.1.1. Support Service

Support Service is the only component that is installed into the customers domain and is responsible for the first step to prepare a signature request from a given document, as well as the last step that involves combining the signed data and the original document into a signed document that is returned to the calling application.

The main reason that the Support Service is installed in the customers domain is that no sensitive data should ever leave their domain. When preparing a signature request the Support Service is creating a cryptographic hash of the document which is included in the data to be signed. The actual document never leaves the customer.

The Support Service does not have any graphical user interface and is not meant to directly interact with end users. Requests are served through a web service API which is typically consumed by the customer web application that the end user is interacting with. In this way the Support Service is transparent to the end user and easily integrates with already existing customer applications, which can then be easily enhanced with signing capabilities.

1.1.2. Frontend Service

Frontend Service is the Internet facing part of the central service that receives requests generated by the support service through the clients web application and redirects call to selected identify provider and request certificates and signature from the backend service.

1.1.3. Backend Service

The backend service is run in a protected environment and uses a HSM to generate keys and signatures.

1.2. Terminology

PKI	Public Key Infrastructure
CMS	Cryptographic Message Syntax
LTA	Long Term Archival
PDF	Portable Document Format
XML	Extensible Markup Language
API	Application Programming Interface
IDP	Identity Provider

2. Integration Guide

2.1. Support Service WebService

This section describes the technical details of the Signature Support Web Service API, and how to use it to implement digital signature workflow in an existing web application.

2.1.1. Integration Overview

1. Understand the signature workflow. Before starting the implementation it is important to get a clear picture of the workflow that is executed when performing digital signatures - following the technical framework specified by Sweden Connect. Take a moment to study the workflow diagram specified in the sub-section *Understanding communication with a federated signature service* in the end of this section.

2. Get hold of an API authentication certificate. In order to access the web service a client certificate is required. Contact CGI Signature Service for more details if you should be authorized to access the API. This is referred to as `client.jks` in the example in step 3 below.

3. Generate Webservice Client. When you have access to the web service it is time to generate the web service client from the given WSDL available in the service locations specified below. The following is an example of how to generate the client for Java.

NOTE	Make sure environment variable JAVA_HOME is pointing to the JDK to use. If using windows specify java home variable as `%JAVA_HOME%` instead of `$JAVA_HOME`. Also make sure that the issuer of the web service SSL Server certificate is trusted and has been added to `$JAVA_HOME/jre/lib/security/cacerts`.

```
java -classpath $JAVA_HOME/lib/tools.jar -Djavax.net.ssl.keyStore=client.jks
-Djavax.net.ssl.keyStorePassword=foo123 com.sun.tools.internal.ws.WsImport
https://support.v2.st.signatureservice.se/signservice-
support/services/v2/signsupport?wsdl -p se.signatureservice.support.api.v2 -s .
```

4. Use the web service client. You are now ready to add the generated classes to the classpath of your application and to use it to perform operations. Please read the section *Operations* for available operations, and the section *Programming Examples* for concrete examples on how to make web service calls.

2.1.2. Service Location

```
https://<support-service>/signservice-support/services/v2/signsupport?wsdl
```

2.1.3. Authentication

Authentication is performed using X509 client certificate. Contact CGI Signature Service for more details if you should be authorized to access the API.

2.1.4. Operations

The following operations exists in the service. Each operation has request and response parameters described as well as faults that might be generated by the call. Description of non-primitive data types can

be found in section *Common Data Types*.

PrepareSignature

The PrepareSignature call is the first call when signing a document. It prepares a EidSignRequest document that should be sent to central signing service to first authenticate the end user and then perform a signature of a generated cryptographic hash value of the related document.

In the call can either the entire document data be sent or a reference to it (when using an archive) the support service will then compute a cryptographic hash value value of the document for central signing. The hash value contains no sensitive data and can therefore be sent to the central service without any document data leave the organisation.

Request

The request contains the following parameters:

Parameter	Data Type	Description
documents	DocumentRequests	A list of DocumentRequest (or DocumentRef if document should be fetched from archive) containing document data about to be signed. If given profile supports counter signatures is the same operation used again.
user	User	The end user that should sign the document.
authenticationServiceId	String	The id of the authentication service used (Usually the entity id of the selected IDP). It is possible to get a list of available authentication services for a given profile using the call GetAuthenticationServices.
consumerURL	String	URL the requesting application uses to receive the response from the central signing service.

Parameter	Data Type	Description
transactionId	String	***(optional)*** The transaction id that should be used if same id should be used in log files within the entire signature flow in both calling application and signature service application. If not set is it auto generated. *Important*: If transaction id is set by the requesting application, it's the requesting applications responsibility to ensure the transaction id is unique. Support service will check that the same transaction id is not used twice as long as a signature flow is active but do not guarantee it is unique after that. If transation id is omitted and generated by support service is always a new transaction id generated for each call to prepareSignature.
profile	String	The related profile used for the signature. A profile defines a configured workflow inside the server.
signMessage	String	***(optional)*** Sign message shown prompted to the end user before signing. If sign message is used is depending on profile configuration.

Response

The response contains the following parameters:

Parameter	Data Type	Description
signRequest	String	The signRequest data in Base64 encoding used in the "EidSignRequest" HTTP POST FORM parameter when redirecting the user to the central signature service.
actionURL	String	The URL to the central signing service the requesting application should redirect the user.

Parameter	Data Type	Description
transactionId	String	The id of the related transaction, is auto generated if not specified in request call. **Important: This must be used as value for relay state when sending request to the signature service, in order to be compliant with ELN-0607:3.1.1**
profile	String	The name of the profile. If no profile was specified will the value DEFAULT be used.

Generated Faults

The following faults are generated, see section Fault Descriptions for details:

- ClientErrorException
- ServerErrorException
- IOErrorException

CompleteSignature

The CompleteSignatureResponse is called after the central signing service returns a EidSignResponse inserting the generated signatures into the document data to generate valid signed documents.

If problems occurred in the central signing service is ClientErrorException or ServerErrorException faults generated depending on the problem.

Request

The request contains the following parameters:

Parameter	Data Type	Description
transactionId	String	The id of the related transaction.
signResponse	String	The signResponse data in Base64 encoding used in the "EidSignResponse" HTTP POST FORM parameter set in response from central signing service.
returnReference	Boolean	***(optional)*** A flag indicating if the full signed document should be returned or just a reference id if document is stored in an archive, (Default: false).

Response

The response contains the following parameters:

Parameter	Data Type	Description
documents	List<Document> *or* List<DocumentRef>	A list of Documents containing signed data or DocumentRefs if only reference id should be returned.

Generated Faults

The following faults are generated, see section Fault Descriptions for details:

- ClientErrorException
- ServerErrorException
- IOErrorException

CancelSignature

The Cancel Signature can be called if the requesting application want to abort a transaction between the PrepareSignature and CompleteSignatureResponse calls. For example due to a timeout or a user logout in requesting application.

Request

The request contains the following parameters:

Parameter	Data Type	Description
transactionId	String	The ID of the transaction to cancel.

Response

The response contains no response parameters.

Generated Faults

The following faults are generated, see section Fault Descriptions for details:

- ClientErrorException
- ServerErrorException
- IOErrorException

VerifyDocument

Call to verify that a single signed document is valid according to a specified profile.

Request

The request contains the following parameters:

Parameter	Data Type	Description
profile	String	The related profile used for the signature. A profile defines a configured workflow inside the server.
document	Document *or* DocumentRef	The Document or DocumentRef to verify. If archive is used and supported by profile is a Document reference sufficient otherwise should a Document be specified.

Response

The response contains the following parameters:

Parameter	Data Type	Description
verifies	Boolean	A boolean flag indicating that the signature is valid.
verificationErrorCode	String	The error code if signature doesn't verify.
verificationErrorMessages	String	Human readable Message containing localized message data that can be displayed to an end user.
signatures	List<Signature>	A list of signatures in the document, See Signature data type for more information.
referenceId	String	The document reference ID.
reportMimeType	String	Mimetype of report in reportData, if available.
reportData	byte[]	Contains validation report in binary format according to reportMimeType.

Generated Faults

The following faults are generated, see section Fault Descriptions for details:

- ClientErrorException
- ServerErrorException
- IOErrorException

VerifyTransaction

Call to verify multiple signed documents related to a transaction in a single call.

Request

The request contains the following parameters:

Parameter	Data Type	Description
profile	String	The related profile used for the signature. A profile defines a configured workflow inside the server.
transactionId	String	*Reserved for future use and can be omitted or set to null.*
documents	List<Document> *or* List<DocumentRef>	A list of documents or document reference to verify.

Response

The response contains the following parameters:

Parameter	Data Type	Description
verifies	Boolean	A boolean flag indicating that all document signatures are valid
verificationErrorCode	String	The error code if one of the signatures doesn't verify.
verificationErrorMessages	String	Human readable Message containing localized message data that can be displayed to an end user.
verifiedDocuments	List<VerifyDocumentResponse>	A list of VerifyDocumentResponse containing details for each verified document *(See VerifyDocument response parameters)*

Generated Faults

The following faults are generated, see section Fault Descriptions for details:

- ClientErrorException
- ServerErrorException
- IOErrorException

GetAuthenticationServices

Method to retrieve configured IDPs for the end user to choose from in the requesting application depending on configured profile.

Request

The request contains the following parameters:

Parameter	Data Type	Description
profile	String	The related profile used. A profile defines a configured workflow inside the server.

Response

The GetAuthenticationServicesResponse contains the list of AuthenticationService, a data structure that contain the following properties:

Parameter	Data Type	Description
authenticationServiceId	String	The id of the authentication service (Usually the entity id of the selected IDP).
displayName	String	Human readable display name of the authentication service.
userIdValidator	String	***(optional)*** A Regular Expression string on how the end user id should be validated.
validationMessages	String	***(optional)*** Message to display to the end user on how the userId format should be specified.

Generated Faults

The following faults are generated, see section Fault Descriptions for details:

- ClientErrorException
- ServerErrorException
- IOErrorException

2.1.5. Common Data Types

This section describes the data types that are common between operations.

DocumentRequestsType

A DocumentRequestsType contains a list of documents and/or document references about to be signed.

Property	Type	Description
documentOrDocumentRef	List<Object>	List containing `DocumentType` and/or `DocumentRefType` describing documents about to be signed.

DocumentType

A DocumentType contains information about a document about to be signed.

Property	Type	Description
type	String	The type of document as MIME type. See section Document Mime Types for details.
data	byte[]	The binary document data in Base64 encoding.
name	String	The name of the document.
referenceId	String	***(optional)*** The reference id of the document as string representing a unique id of the document. If not specified it will be generated by the service.

DocumentRefType

Data type that only contain the reference to a document used for application integrated with an archive and document data shouldn't be sent from server to requester.

Property	Type	Description
referenceId	String	Document reference ID.

DocumentResponsesType

A DocumentResponsesType contains a list of signed documents and/or document references signed by the service.

Property	Type	Description
documentOrDocumentRef	List<Object>	List containing `DocumentResponseType` and/or `DocumentRefType` describing documents signed by the service.

DocumentResponseType

A DocumentResponseType contains information about a signed document.

Property	Type	Description
type	String	The type of document as MIME type. See section Document Mime Types for details.
data	byte[]	The binary document data in Base64 encoding.

Property	Type	Description
name	String	The name of the document.
referenceId	String	***(optional)*** The reference id of the document as string representing a unique id of the document. If not specified it will be generated by the service.
signatures	SignersType	Signature data
hasDetachedSignature	boolean	Describes if detached signature data is available.
detachedSignatureData	byte[]	Binary detached signature data in Base64 encoding.
validationInfo	VerifyDocumentResponseType	***(optional)*** Document validation information, if available.

SignersType

A SignersType contains a list of signatures related to a signed document.

Property	Type	Description
signatures	List<SignatureType>	List of signatures related to a signed document.

SignatureType

A data structure providing information about a signature in a document

Property	Type	Description
signerId	String	Unique id of the signer. Can be personal number or username depending on environment.
signerDisplayName	String	Displayable name of the signer.
signerCertificate	byte[]	The signing X509 Certificate in Base64 DER encoding.
issuerId	String	The id of the issuer of the certificate, usually subject DN of issuing CA.
signingAlgorithm	String	The algorithm used when signing of the document.
validFrom	XMLGregorianCalendar	The date the signature (i.e signing certificate) is valid from.

Property	Type	Description
validTo	XMLGregorianCalendar	The date the signature (i.e signing certificate) will expire.
levelOfAssurance	String	The level of assurance of the original authentication mechanism use to perform the signature (LOA).

VerifyDocumentResponseType

Data type containing information about a verified document.

Property	Type	Description
verifies	boolean	Indicates if verification was successfull (true) or not (false).
verificationErrorCode	Integer	Indication of verification result. Possible values are 0=TOTAL_PASSED, 1=PASSED, 2=INDETERMINATE, 3=FAILED, 4=TOTAL_FAILED
verificationErrorMessages	MessagesType	Messages that gives information about the verification result.
signatures	SignersType	List of Signatures related to the verified document.
referenceId	String	Reference ID of verified document.
reportMimeType	String	Validation report mime type.
reportData	byte[]	Validation report data depending on mime type.

MessagesType

A MessagesType holds a list of messages

Property	Type	Description
message	List<MessageType>	List of messages.

MessageType

A data type that contains a localized message for a given language.

Property	Type	Description
lang	String	The language code of the related message (Attribute). Should be a value according to IETF BCP 47 language code format, for example: 'en' or 'en_UK'.
text	String	The message string in specified language.

UserType

A data type that specifies the ID and meta data about the end user that should sign a document.

Property	Type	Description
userId	String	The ID of the end user. Can be personal number or username depending on environment.
role	String	***(optional)*** The role the user signs as. A value defined for each profile. Default profile does not specify any roles and can be omitted.
userAttributes	List<AttributeType>	***(optional)*** A list of key, value attributes to provide extra information about a User that cannot be fetched by the authentication service. Valid values is depending on specified profile.

AttributeType

A data type that holds a key/value attribute pair.

Property	Type	Description
key	String	Attribute key.
value	String	Attribute value.

2.1.6. Document Mime Types

When sending documents to the service the document type must be specified. This should be specified as a mimetype string and will affect the type of signature that is going to be created. The following table gives an overview of the different mimetypes and how they are mapped with signature types.

Type (Mime Type)	Signature Type
text/xml	XML/XAdES

Type (Mime Type)	Signature Type
application/pdf	PDF/PAdES
application/octet-stream *(or any other)*	CMS/CAdES

2.1.7. Fault Descriptions

This section describes the possible fault messages that can be generated by the available operations. All faults have the same data structure with the following properties:

Property	Description
code	The defined error code of the error. See section Error Code Reference for details.
messages	A list of Message with error message in each configured language that can be displayed for the end user.
detailMessage	A technical error message that requesting application can use in log files for a more detailed description of what went wrong.

ClientErrorException

ClientErrorException fault is generated due to invalid input by the requesting application.

IOErrorException

IOErrorException fault is generated when communication problems such as time-out occurred and might be resolved by trying regenerate the request call.

ServerErrorException

ServerErrorException fault is generated due internal problem in the service that needs to be resolved, either the support service or central systems.

2.1.8. Java Example

The following imports are used within the Java programming examples.

```
// Standard Java
import javax.xml.namespace.QName;
import java.io.FileInputStream;
import java.net.MalformedURLException;
import java.net.URL;

// Generated from WSDL
import se.signatureservice.support.api.v2.*;

// Apache Commons I/O (https://commons.apache.org/io)
import org.apache.commons.io.IOUtils;
```

STEP 1. Initialize Web Service Client

```
SignSupport supportService;
try {
    URL url = new URL(wsdlLocation);
    QName qName = new QName(
        "http://signsupport.v2.api.support.signatureservice.se/",
        "SignSupportV2Endpoint"
    );
    supportService = new SignSupportV2Endpoint(url, qName).getSignSupportV2Port();
} catch(MalformedURLException e){
    // Log error
}
```

Step 2. Prepare Document for Signature

In order to sign the document, the first step is to prepare the document for the signature workflow. To do this the requesting application should perform a call to the `prepareSignature` method.

The following example shows how this can be done.

```
// User that should sign document.
String userId = "196707183130";

// Document to sign.
String documentName = "testdocument.xml";
String documentType = "text/xml";
byte[] documentData = IOUtils.toByteArray(new
FileInputStream("resources/testdocument.xml"));

// Metadata entity ID of IdP to request authentication from.
String authenticationServiceId =
"https://idp.v2.st.signatureservice.se/samlv2/idp/metadata";
```

```
// The consumer URL within application to redirect the user to with the Sign Response.
String consumerURL = "http://localhost/response";

// Profile to use for signature.
String signatureProfile = "Profile_ECDSA_Enveloped_BaselineB";

// Factory to create web service objects
ObjectFactory objectFactory = new ObjectFactory();

UserType user = objectFactory.createUserType();
user.setUserId(userId);

DocumentType document = objectFactory.createDocumentType();
document.setName(documentName);
document.setType(documentType);
document.setData(documentData);

DocumentRequestsType documents = objectFactory.createDocumentRequestsType();
documents.getDocumentOrDocumentRef().add(document);


// Perform call to Support Service.
PreparedSignatureResponseType preparedSignature  = supportService.prepareSignature(
        null, // Auto-generated transaction ID
        signatureProfile,
        documents,
        null, // No sign message
        user,
        authenticationServiceId,
        consumerURL);

// Now it is possible to extract the generated transaction ID and SignRequest
document.
String transactionId = preparedSignature.getTransactionId();
String signRequest = preparedSignature.getSignRequest();
String actionURL = preparedSignature.getActionURL();
```

Step 3. Redirect the end user to the central signing service.

Next step for the requesting application is to generate a HTML form for the end user that redirects him to the central signing service. An example on such a form is specified below. replace the three placeholders `INSERT_TRANSACTIONID`, `INSERT_SIGNREQUEST` and `INSERT_ACTIONURL` with values extracted in step 2 before rendering to user browser.

```
<html>
<body onload='document.forms[0].submit()'>
    <noscript>
        <p>
            <strong>Note:</strong>
            Since your browser does not support JavaScript,
            you must press the Continue button once to
            proceed.
        </p>
    </noscript>
    <form action='INSERT_ACTIONURL' method='post'>
        <div>
            <input name='Binding' type='hidden' value='POST/XML/1.0'>
            <input name='RelayState' type='hidden' value='INSERT_TRANSACTIONID'>
            <input name='EidSignRequest' type='hidden' value='INSERT_SIGNREQUEST'>
        </div>
        <noscript>
            <div>
                <input type='submit' value='Continue'>
            </div>
        </noscript>
    </form>
</body>
</html>
```

Step 4. Recieve Response on ConsumerURL and Complete the Signature.

The requesting application should have controller/servlet waiting for response from the central signing service that will POST the following parameters to the specified consumer URL:

- RelayState : The same relay state as specified in Step 2, which is equal to the transactionId.
- EidSignResponse : The EidSignResponse data that should be sent to support service.

After extracting the parameters from the received form the following code gives an example on how to complete the signature in order to obtain the signed document.

```
CompleteSignatureResponseType completeSignature =
    supportService.completeSignature(signResponse, relayState, false);

// Get signed document from response. In this example we know that we can expect
// a single signed document. In real world we should handle zero-or-many signed
// documents in the result.
DocumentResponseType signedDocument =
    (DocumentResponseType)completeSignature.getDocuments()
        .getDocumentOrDocumentRef()
        .get(0);
```

Step 5. Verify the document using the web service (Optional)

The document can be sent to the support service for verification and to retrieve a validation report. The following example shows how this can be done, following the example code above.

```
DocumentResponseChoiceType documentChoice =
    objectFactory.createDocumentResponseChoiceType();
documentChoice.setDocument(signedDocument);
VerifyDocumentResponseType verifyResponse =
    supportService.verifyDocument(signatureProfile, documentChoice);

if(verifyResponse.isVerifies()){
    // Document verified successfully!
} else {
    // Document failed to verify!
}

// The verification response also contains a report that can be displayed.
// Implementation of displayReport is left as an exercise to the reader.
displayReport(verifyResponse.getReportMimeType(), verifyResponse.getReportData());
```

2.1.9. C# Example

STEP 1. Initialize Web Service Client

Using Visual Studio add a reference to the Support Service:

1. Right-click on project references and choose **Add Service Reference…**.
2. Enter URL to service (ex. **https://supportservice.customer.com/signservice-support/services/v2/signsupport?wsdl**).
3. Choose a name for the namespace (ex. **SignServiceSupport**).
4. Click on **OK**.

Once the reference is in place, an instance of the client can be created:

```
 SignSupportClient supportClient = new SignSupportClient();
```

Step 2. Prepare Document for Signature

```
// Create user.
UserType user = new UserType {
    userId = "196707183130",
    userAttributes = new AttributeType[] {
        new AttributeType() {
            key = "name",
            value = "Test User"
        }
```

```
    }
};

// Create document.
DocumentRequestsType documentRequest = new DocumentRequestsType {
    Items = new DocumentType[] {
        new DocumentType {
            name = "testdocument.xml",
            data = Encoding.UTF8.GetBytes("<document><test
id=\"42\">123</test></document>"),
            type = "text/xml"
        }
    }
};

// Profile to use. Will also be needed during optional verification.
string signatureProfile = "Profile_ECDSA_Enveloped_BaselineB";

// Call Support Service in order to prepare the signature.
PreparedSignatureResponseType preparedSignature = supportClient.PrepareSignature(
    // Transaction ID parameter. If not specified it will be generated by the service.
    null,

    // Profile to use. Must be a valid profile name defined by the service.
    signatureProfile,

    // Documents to sign.
    documentRequest,

    // Sign message.
    "Do you want to sign xyz?",

    // User details.
    user,

    // authentication service (IdP) to request. Available services are
    // defined by the service.
    "https://m00-mg-local.idpst.funktionstjanster.se/samlv2/idp/metadata/6/7",

    // Consumer URL where the user will be redirected to when the signature
    // response has been received. Must be authorized by the service.
    // This controller should extract EidSignResponse and RelayState from
    // request parameters and call the service to complete signature and to
    // receive the signed documents.
    "https://app.customer.com/process"
);

// Now it is possible to extract the generated transaction ID and SignRequest
document.
```

```
string transactionId = preparedSignature.transactionId;
string signRequest = preparedSignature.signRequest;
string actionURL = preparedSignature.actionURL;
```

Step 3. Redirect the end user to the central signing service.

Next step for the requesting application is to generate a HTML form for the end user that redirects him to the central signing service. An example on such a form is specified below. replace the three placeholders `INSERT_TRANSACTIONID`, `INSERT_SIGNREQUEST` and `INSERT_ACTIONURL` with values extracted in step 2 before rendering to user browser.

```
<html>
<body onload='document.forms[0].submit()'>
    <noscript>
        <p>
            <strong>Note:</strong>
            Since your browser does not support JavaScript,
            you must press the Continue button once to
            proceed.
        </p>
    </noscript>
    <form action='INSERT_ACTIONURL' method='post'>
        <div>
            <input name='Binding' type='hidden' value='POST/XML/1.0'>
            <input name='RelayState' type='hidden' value='INSERT_TRANSACTIONID'>
            <input name='EidSignRequest' type='hidden' value='INSERT_SIGNREQUEST'>
        </div>
        <noscript>
            <div>
                <input type='submit' value='Continue'>
            </div>
        </noscript>
    </form>
</body>
</html>
```

Step 4. Recieve Response on ConsumerURL and Complete the Signature.

The requesting application should have controller/servlet waiting for response from the central signing service that will POST the following parameters to the specified consumer URL:

- RelayState : The same relay state as specified in Step 2, which is equal to the transactionId.
- EidSignResponse : The EidSignResponse data that should be sent to support service.

After extracting the parameters from the received form the following code gives an example on how to complete the signature in order to obtain the signed document.

```
CompleteSignatureResponseType completeSignature = supportClient.CompleteSignature(
    signResponse, relayState, false
);

// Get signed document from response. In this example we know that we can expect
// a single signed document. In real world we should handle zero-or-many signed
// documents in the result.
DocumentResponseType signedDocument =
(DocumentResponseType)completeSignature.documents.Items[0];
```

Step 5. Verify the document using the web service (Optional)

```
DocumentResponseChoiceType documentChoice = new DocumentResponseChoiceType();
documentChoice.Item = signedDocument;
VerifyDocumentResponseType verifyResponse =
supportClient.VerifyDocument(signatureProfile, documentChoice);

if(verifyResponse.verifies)
{
    // Document verified successfully!
}
else
{
    // Document failed to verify!
}

// The verification response also contains a report that can be displayed.
// Implementation of displayReport is left as an exercise to the reader.
displayReport(verifyResponse.reportMimeType, verifyResponse.reportData);
```

2.1.10. WSDL Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://signsupport.v2.api.support.signatureservice.se/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="SignSupportV2Endpoint"
targetNamespace="http://signsupport.v2.api.support.signatureservice.se/">
  <wsdl:documentation>SignSupport Service V2 contains help methods to prepare,
finalize and verify digital signatures for web services.</wsdl:documentation>
  <wsdl:types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="v2.api.support.signatureservice.se"
xmlns:api="v2.api.support.signatureservice.se" elementFormDefault="qualified"
```

```
targetNamespace="v2.api.support.signatureservice.se" version="1.0">
      <xs:complexType name="MessagesType">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" name="message" type="tns:MessageType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="MessageType">
        <xs:sequence>
          <xs:element name="text" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="lang" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:complexType name="DocumentResponseChoiceType">
        <xs:choice>
          <xs:element name="document" type="tns:DocumentResponseType"/>
          <xs:element name="documentRef" type="tns:DocumentRefType"/>
        </xs:choice>
      </xs:complexType>
      <xs:complexType name="DocumentResponseType">
        <xs:complexContent>
          <xs:extension base="tns:AbstractDocumentType">
            <xs:sequence>
              <xs:element minOccurs="0" name="signatures" type="tns:SignersType"/>
              <xs:element default="false" name="hasDetachedSignature"
type="xs:boolean"/>
              <xs:element minOccurs="0" name="detachedSignatureData"
type="xs:base64Binary"/>
              <xs:element minOccurs="0" name="validationInfo"
type="tns:VerifyDocumentResponseType"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="AbstractDocumentType">
        <xs:sequence>
          <xs:element name="type" type="xs:string"/>
          <xs:element name="data" type="xs:base64Binary"/>
          <xs:element name="name" type="xs:string"/>
          <xs:element minOccurs="0" name="referenceId" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="SignersType">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" name="signatures"
type="tns:SignatureType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="SignatureType">
        <xs:sequence>
```

```
          <xs:element name="signerId" type="xs:string"/>
          <xs:element name="signerDisplayName" type="xs:string"/>
          <xs:element name="signerCertificate" type="xs:base64Binary"/>
          <xs:element name="issuerId" type="xs:string"/>
          <xs:element name="signingAlgorithm" type="xs:string"/>
          <xs:element name="signingDate" type="xs:dateTime"/>
          <xs:element name="validFrom" type="xs:dateTime"/>
          <xs:element name="validTo" type="xs:dateTime"/>
          <xs:element name="levelOfAssurance" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="VerifyDocumentResponseType">
        <xs:complexContent>
          <xs:extension base="tns:AbstractVerifiedType">
            <xs:sequence>
              <xs:element minOccurs="0" name="signatures" type="tns:SignersType"/>
              <xs:element name="referenceId" type="xs:string"/>
              <xs:element minOccurs="0" name="reportMimeType" type="xs:string"/>
              <xs:element minOccurs="0" name="reportData" type="xs:base64Binary"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="AbstractVerifiedType">
        <xs:sequence>
          <xs:element name="verifies" type="xs:boolean"/>
          <xs:element minOccurs="0" name="verificationErrorCode" type="xs:int"/>
          <xs:element minOccurs="0" name="verificationErrorMessages"
type="tns:MessagesType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="DocumentRefType">
        <xs:sequence>
          <xs:element name="referenceId" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="VerifyTransactionResponseType">
        <xs:complexContent>
          <xs:extension base="tns:AbstractVerifiedType">
            <xs:sequence>
              <xs:element minOccurs="0" name="transactionId" type="xs:string"/>
              <xs:element name="verifiedDocuments" type="tns:VerifiedDocumentsType"/>
            </xs:sequence>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="VerifiedDocumentsType">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" name="verifiedDocuments"
```

```
type="tns:VerifyDocumentResponseType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="AuthenticationServiceType">
        <xs:sequence>
          <xs:element name="authenticationServiceId" type="xs:string"/>
          <xs:element name="displayName" type="xs:string"/>
          <xs:element minOccurs="0" name="userIdValidator" type="xs:string"/>
          <xs:element minOccurs="0" name="validationMessages"
type="tns:MessagesType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="CompleteSignatureResponseType">
        <xs:sequence>
          <xs:element name="documents" type="tns:DocumentResponsesType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="DocumentResponsesType">
        <xs:sequence>
          <xs:choice maxOccurs="unbounded">
            <xs:element name="document" type="tns:DocumentResponseType"/>
            <xs:element name="documentRef" type="tns:DocumentRefType"/>
          </xs:choice>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="DocumentRequestsType">
        <xs:sequence>
          <xs:choice maxOccurs="unbounded">
            <xs:element name="document" type="tns:DocumentType"/>
            <xs:element name="documentRef" type="tns:DocumentRefType"/>
          </xs:choice>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="DocumentType">
        <xs:complexContent>
          <xs:extension base="tns:AbstractDocumentType">
            <xs:all/>
          </xs:extension>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="UserType">
        <xs:sequence>
          <xs:element name="userId" type="xs:string"/>
          <xs:element minOccurs="0" name="role" type="xs:string"/>
          <xs:element maxOccurs="unbounded" minOccurs="0" name="userAttributes"
type="tns:AttributeType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="AttributeType">
```

```
        <xs:sequence>
          <xs:element name="key" type="xs:string"/>
          <xs:element name="value" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="PreparedSignatureResponseType">
        <xs:sequence>
          <xs:element name="signRequest" type="xs:string"/>
          <xs:element name="actionURL" type="xs:string"/>
          <xs:element name="transactionId" type="xs:string"/>
          <xs:element name="profile" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://signsupport.v2.api.support.signatureservice.se/"
xmlns:ns1="v2.api.support.signatureservice.se" attributeFormDefault="unqualified"
elementFormDefault="unqualified"
targetNamespace="http://signsupport.v2.api.support.signatureservice.se/">
      <xs:import namespace="v2.api.support.signatureservice.se"/>
      <xs:element name="CancelSignature" type="tns:CancelSignature"/>
      <xs:element name="CancelSignatureResponse" type="tns:CancelSignatureResponse"/>
      <xs:element name="CompleteSignature" type="tns:CompleteSignature"/>
      <xs:element name="CompleteSignatureResponse"
type="tns:CompleteSignatureResponse"/>
      <xs:element name="GetAuthenticationServices"
type="tns:GetAuthenticationServices"/>
      <xs:element name="GetAuthenticationServicesResponse"
type="tns:GetAuthenticationServicesResponse"/>
      <xs:element name="PrepareSignature" type="tns:PrepareSignature"/>
      <xs:element name="PrepareSignatureResponse"
type="tns:PrepareSignatureResponse"/>
      <xs:element name="VerifyDocument" type="tns:VerifyDocument"/>
      <xs:element name="VerifyDocumentResponse" type="tns:VerifyDocumentResponse"/>
      <xs:element name="VerifyTransaction" type="tns:VerifyTransaction"/>
      <xs:element name="VerifyTransactionResponse"
type="tns:VerifyTransactionResponse"/>
      <xs:complexType name="CancelSignature">
        <xs:sequence>
          <xs:element name="transactionId" type="xs:string"/>
          <xs:element minOccurs="0" name="reason" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="CancelSignatureResponse">
        <xs:sequence/>
      </xs:complexType>
      <xs:complexType name="VerifyTransaction">
        <xs:sequence>
          <xs:element minOccurs="0" name="profile" type="xs:string"/>
```

```
          <xs:element minOccurs="0" name="transactionId" type="xs:string"/>
          <xs:element maxOccurs="unbounded" minOccurs="0" name="documents"
type="ns1:DocumentResponseChoiceType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="VerifyTransactionResponse">
        <xs:sequence>
          <xs:element name="VerifyTransactionResponse"
type="ns1:VerifyTransactionResponseType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetAuthenticationServices">
        <xs:sequence>
          <xs:element minOccurs="0" name="profile" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="GetAuthenticationServicesResponse">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" name="GetAuthenticationServicesResponse"
type="ns1:AuthenticationServiceType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="CompleteSignature">
        <xs:sequence>
          <xs:element name="signResponse" type="xs:string"/>
          <xs:element name="transactionId" type="xs:string"/>
          <xs:element default="false" name="returnReference" type="xs:boolean"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="CompleteSignatureResponse">
        <xs:sequence>
          <xs:element name="CompleteSignatureResponse"
type="ns1:CompleteSignatureResponseType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="PrepareSignature">
        <xs:sequence>
          <xs:element minOccurs="0" name="transactionId" type="xs:string"/>
          <xs:element minOccurs="0" name="profile" type="xs:string"/>
          <xs:element name="documents" type="ns1:DocumentRequestsType"/>
          <xs:element minOccurs="0" name="signMessage" type="xs:string"/>
          <xs:element name="user" type="ns1:UserType"/>
          <xs:element name="authenticationServiceId" type="xs:string"/>
          <xs:element name="consumerURL" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="PrepareSignatureResponse">
        <xs:sequence>
          <xs:element name="PrepareSignatureResponse"
```

```
type="ns1:PreparedSignatureResponseType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="VerifyDocument">
        <xs:sequence>
          <xs:element minOccurs="0" name="profile" type="xs:string"/>
          <xs:element name="document" type="ns1:DocumentResponseChoiceType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="VerifyDocumentResponse">
        <xs:sequence>
          <xs:element name="VerifyDocumentResponse"
type="ns1:VerifyDocumentResponseType"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="IOErrorException" type="tns:IOErrorException"/>
      <xs:complexType name="IOErrorException">
        <xs:sequence>
          <xs:element name="code" type="xs:string"/>
          <xs:element name="messages" type="ns1:MessagesType"/>
          <xs:element name="detailMessage" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="ClientErrorException" type="tns:ClientErrorException"/>
      <xs:complexType name="ClientErrorException">
        <xs:sequence>
          <xs:element name="code" type="xs:string"/>
          <xs:element name="messages" type="ns1:MessagesType"/>
          <xs:element name="detailMessage" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="ServerErrorException" type="tns:ServerErrorException"/>
      <xs:complexType name="ServerErrorException">
        <xs:sequence>
          <xs:element name="code" type="xs:string"/>
          <xs:element name="messages" type="ns1:MessagesType"/>
          <xs:element name="detailMessage" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="CompleteSignatureResponse">
    <wsdl:part element="tns:CompleteSignatureResponse" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="PrepareSignature">
    <wsdl:part element="tns:PrepareSignature" name="parameters">
    </wsdl:part>
  </wsdl:message>
```

```
<wsdl:message name="ClientErrorException">
  <wsdl:part element="tns:ClientErrorException" name="ClientErrorException">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="PrepareSignatureResponse">
  <wsdl:part element="tns:PrepareSignatureResponse" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="VerifyDocumentResponse">
  <wsdl:part element="tns:VerifyDocumentResponse" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="CancelSignatureResponse">
  <wsdl:part element="tns:CancelSignatureResponse" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="ServerErrorException">
  <wsdl:part element="tns:ServerErrorException" name="ServerErrorException">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="VerifyDocument">
  <wsdl:part element="tns:VerifyDocument" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="IOErrorException">
  <wsdl:part element="tns:IOErrorException" name="IOErrorException">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="CancelSignature">
  <wsdl:part element="tns:CancelSignature" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="VerifyTransaction">
  <wsdl:part element="tns:VerifyTransaction" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="VerifyTransactionResponse">
  <wsdl:part element="tns:VerifyTransactionResponse" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="GetAuthenticationServices">
  <wsdl:part element="tns:GetAuthenticationServices" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="GetAuthenticationServicesResponse">
  <wsdl:part element="tns:GetAuthenticationServicesResponse" name="parameters">
  </wsdl:part>
</wsdl:message>
<wsdl:message name="CompleteSignature">
```

```
    <wsdl:part element="tns:CompleteSignature" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="SignSupport">
    <wsdl:operation name="CancelSignature">
      <wsdl:documentation>The Cancel Signature can be called if the requesting
application want to abort a transaction between the
PrepareSignature and CompleteSignatureResponse calls. For example due to a timeout or
a user logout in requesting
application.</wsdl:documentation>
      <wsdl:input message="tns:CancelSignature" name="CancelSignature">
    </wsdl:input>
      <wsdl:output message="tns:CancelSignatureResponse"
name="CancelSignatureResponse">
    </wsdl:output>
      <wsdl:fault message="tns:IOErrorException" name="IOErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ClientErrorException" name="ClientErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ServerErrorException" name="ServerErrorException">
    </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="VerifyTransaction">
      <wsdl:documentation>Call to verify one or more documents related to an
transaction.</wsdl:documentation>
      <wsdl:input message="tns:VerifyTransaction" name="VerifyTransaction">
    </wsdl:input>
      <wsdl:output message="tns:VerifyTransactionResponse"
name="VerifyTransactionResponse">
    </wsdl:output>
      <wsdl:fault message="tns:IOErrorException" name="IOErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ClientErrorException" name="ClientErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ServerErrorException" name="ServerErrorException">
    </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="GetAuthenticationServices">
      <wsdl:documentation>Method to retrieve configured IDPs for the end user to
choose from in the requesting application depending
                    on configured profile..</wsdl:documentation>
      <wsdl:input message="tns:GetAuthenticationServices"
name="GetAuthenticationServices">
    </wsdl:input>
      <wsdl:output message="tns:GetAuthenticationServicesResponse"
name="GetAuthenticationServicesResponse">
    </wsdl:output>
      <wsdl:fault message="tns:IOErrorException" name="IOErrorException">
    </wsdl:fault>
```

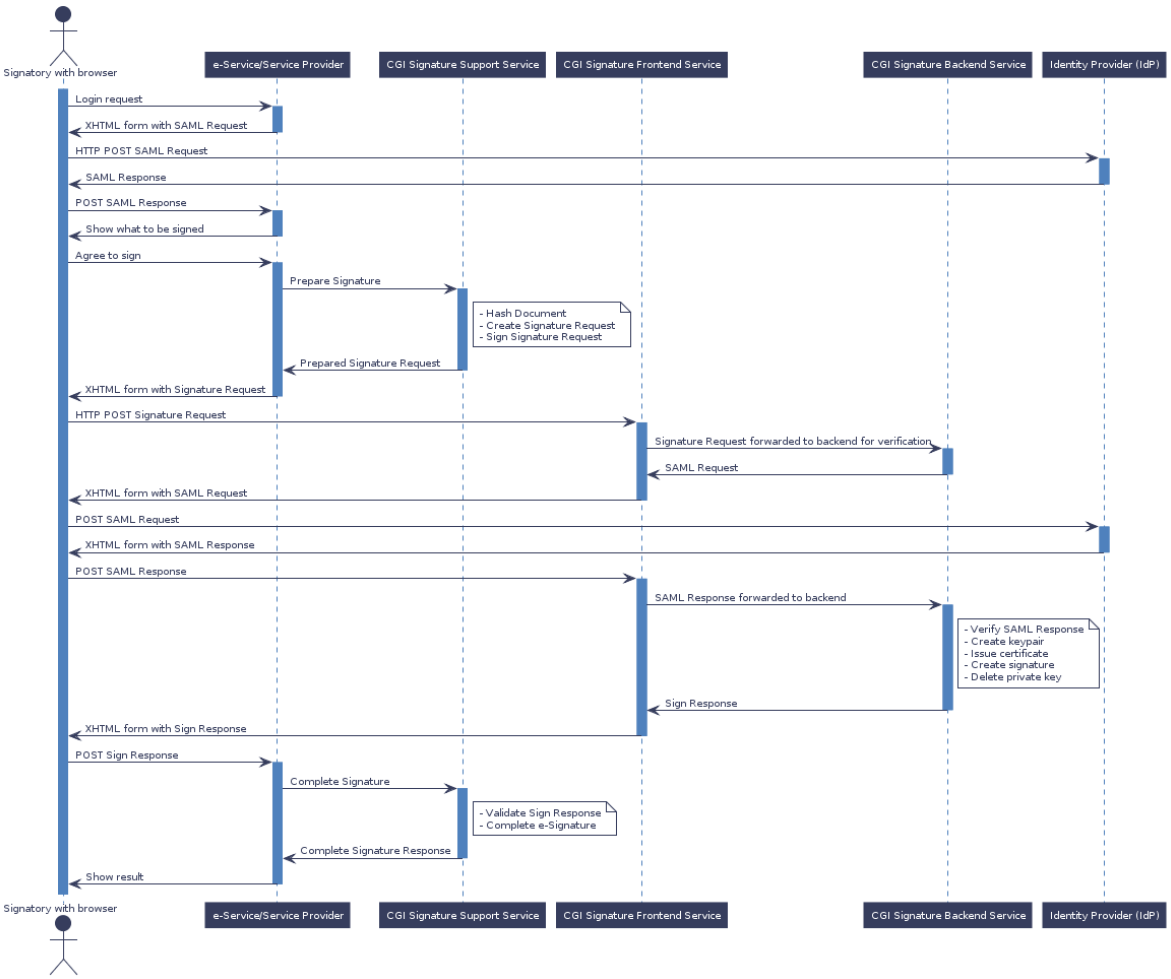
```
      <wsdl:fault message="tns:ClientErrorException" name="ClientErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ServerErrorException" name="ServerErrorException">
    </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="CompleteSignature">
      <wsdl:documentation>The CompleteSignatureResponse is called after the central
signing service returns a EidSignResponse inserting the
generated signatures into the document data to generate valid signed documents.

If problems occurred in the central signing service is ClientErrorException or
ServerErrorException faults generated
depending on the problem.</wsdl:documentation>
      <wsdl:input message="tns:CompleteSignature" name="CompleteSignature">
    </wsdl:input>
      <wsdl:output message="tns:CompleteSignatureResponse"
name="CompleteSignatureResponse">
    </wsdl:output>
      <wsdl:fault message="tns:IOErrorException" name="IOErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ClientErrorException" name="ClientErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ServerErrorException" name="ServerErrorException">
    </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="PrepareSignature">
      <wsdl:documentation>The PrepareSignature call is the first call when signing a
document. It prepares a EidSignRequest document that should
be sent to central signing service to first authenticate the end user and then perform
a signature of a generated
cryptographic hash value of the related document.

In the call can either the entire document data be sent or a reference to it (when
using an archive) the support service
will then compute a cryptographic hash value value of the document for central
signing. The hash value contains no
sensitive data and can therefore be sent to the central service without any document
data leave the organisation.</wsdl:documentation>
      <wsdl:input message="tns:PrepareSignature" name="PrepareSignature">
    </wsdl:input>
      <wsdl:output message="tns:PrepareSignatureResponse"
name="PrepareSignatureResponse">
    </wsdl:output>
      <wsdl:fault message="tns:IOErrorException" name="IOErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ClientErrorException" name="ClientErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ServerErrorException" name="ServerErrorException">
    </wsdl:fault>
```

```
    </wsdl:operation>
    <wsdl:operation name="VerifyDocument">
      <wsdl:documentation>Call to verify that a signed document is valid according to
a specified profile.</wsdl:documentation>
      <wsdl:input message="tns:VerifyDocument" name="VerifyDocument">
    </wsdl:input>
      <wsdl:output message="tns:VerifyDocumentResponse" name="VerifyDocumentResponse">
    </wsdl:output>
      <wsdl:fault message="tns:IOErrorException" name="IOErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ClientErrorException" name="ClientErrorException">
    </wsdl:fault>
      <wsdl:fault message="tns:ServerErrorException" name="ServerErrorException">
    </wsdl:fault>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SignSupportV2EndpointSoapBinding" type="tns:SignSupport">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="CancelSignature">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="CancelSignature">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CancelSignatureResponse">
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="IOErrorException">
        <soap:fault name="IOErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientErrorException">
        <soap:fault name="ClientErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ServerErrorException">
        <soap:fault name="ServerErrorException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="VerifyTransaction">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="VerifyTransaction">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="VerifyTransactionResponse">
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="IOErrorException">
        <soap:fault name="IOErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientErrorException">
        <soap:fault name="ClientErrorException" use="literal"/>
```

```
      </wsdl:fault>
      <wsdl:fault name="ServerErrorException">
        <soap:fault name="ServerErrorException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="GetAuthenticationServices">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="GetAuthenticationServices">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="GetAuthenticationServicesResponse">
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="IOErrorException">
        <soap:fault name="IOErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientErrorException">
        <soap:fault name="ClientErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ServerErrorException">
        <soap:fault name="ServerErrorException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="CompleteSignature">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="CompleteSignature">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="CompleteSignatureResponse">
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="IOErrorException">
        <soap:fault name="IOErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientErrorException">
        <soap:fault name="ClientErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ServerErrorException">
        <soap:fault name="ServerErrorException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="PrepareSignature">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="PrepareSignature">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="PrepareSignatureResponse">
        <soap:body use="literal"/>
      </wsdl:output>
```

```
      <wsdl:fault name="IOErrorException">
        <soap:fault name="IOErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientErrorException">
        <soap:fault name="ClientErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ServerErrorException">
        <soap:fault name="ServerErrorException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="VerifyDocument">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="VerifyDocument">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="VerifyDocumentResponse">
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="IOErrorException">
        <soap:fault name="IOErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ClientErrorException">
        <soap:fault name="ClientErrorException" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="ServerErrorException">
        <soap:fault name="ServerErrorException" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SignSupportV2Endpoint">
    <wsdl:port binding="tns:SignSupportV2EndpointSoapBinding"
name="SignSupportV2Port">
      <soap:address location="http://localhost:9090/signservice-
support/services/v2/signsupport"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

2.1.11. Understanding communication with a federated signature service



Appendix A: Error Codes

Error Codes Support Service

The following table shows the different error codes that can be returned when calling the Signature Support Service API.

Code	Type	Description
10001	Server Error	Required configuration is missing
10002	Server Error	Invalid configuration
10003	Client Error	Unsupported algorithm
10004	Server Error	Invalid digest size
10005	Server Error	Signature certificate chain is missing or invalid
10006	Client Error	Signature type is not supported
10007	Client Error	Invalid profile

Code	Type	Description
10008	Server Error	Error while accessing or processing metadata
10009	Client Error	Unauthorized consumer", ClientErrorException
10010	Client Error	Transaction ID does not meet complexity requiremnets
10011	Client Error	Unknown transaction
10012	Server Error	Error while processing signature response
10013	Client Error	Unsupported operation requested
10014	Server Error	Internal error while processing transaction
10015	Client Error	Unauthorized authentication service
10016	Client Error	Document to be signed is invalid, missing or corrupt
10017	Client Error	Invalid mime type. See documentation for valid mime types
10018	Client Error	SignResponse is missing, invalid or corrupt
10019	Server Error	Error while generating signature request
10020	Server Error	Error while verifying document

Appendix B: Troubleshooting

Troubleshooting Support Service

Symptom	Possible Cause	Fix
Signature creation failed	Time not synchronized	It is very important that the time is synchronized between all different servers and components in the system. Make sure that the time is synchronized using NTP.

Appendix C: Known Problems

This section contains a list of known problems and possible workarounds.

Problem	Description/Workaround
Out of memory problem	Currently the in-memory cache is not cleaned up after each transaction.