Dokumentnamn	**Dokumenttyp**	**Datum**
Eid 2.0 Sign Support Deployment Manual	Manual	2013-05-20
Upprättad av	**Godkänd av**	**Version**
Stefan Santesson		0.3

Eid 2.0 SignSupport Deployment Manual

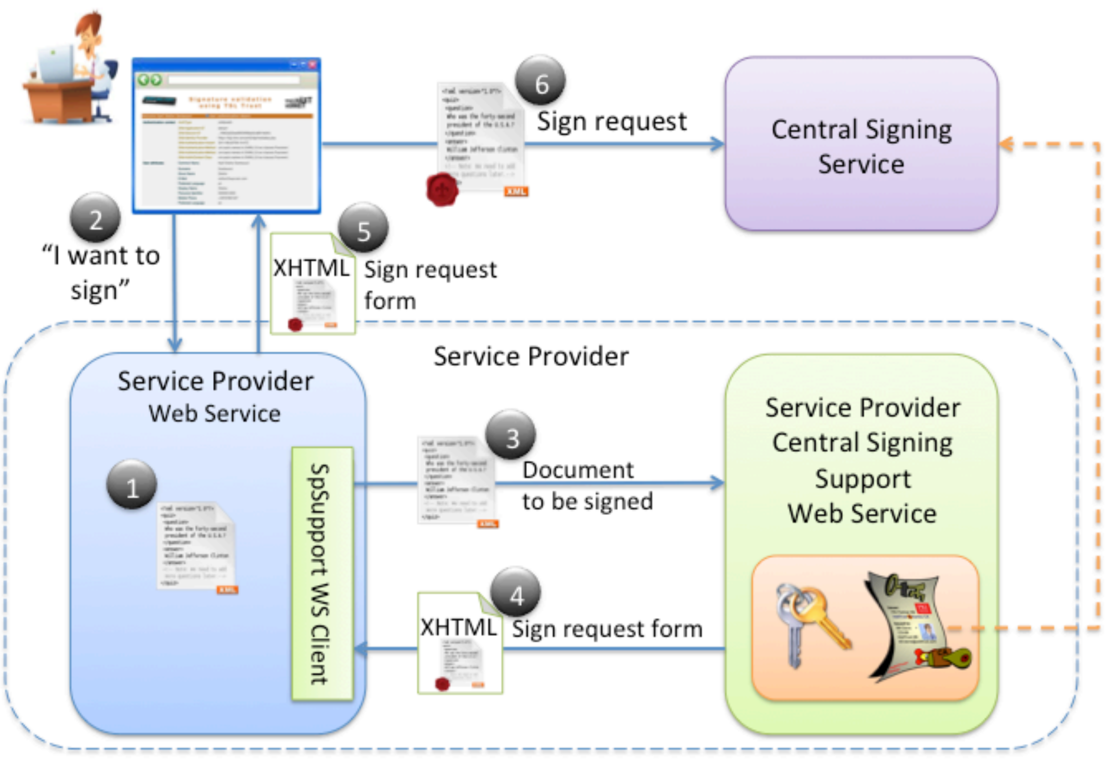

Table of content

1 Overview

It is essential to follow the installation and deployment instructions in this document in order to run the provided services. Failing to do so may prevent services from running properly.

The sign support service provide functionality for signing documents using a deployed central signing service within Testbädden for Eid 2.0

This service is accessible through a Web Services API that is specified through a WSDL, available from the service at:

{path to service}/SpSupportWs?wsdl

To support legacy implementations of this support service, a Java API is also available. However, support for this API may be phased out.

1.1 Server components

Signature validation service

CsSpSupport.war	Sign support service

2 Installation and deployment

2.1 Generic

Deployment and configuration of the signature support service should be done in the following logical order:

1. Configure the application server where the support service is to be deployed (Provided examples describes procedures for setting up a Tomcat server)
2. Deploy the signature support service and start it.
3. Configure the configuration files of the support service.
4. Stop and restart the web service.
5. Export the signature certificate created by the support service and send it to be installed in the central signing service.

More elaborate details are provided in the following subsections.

2.1.1 Tomcat server configuration

The following configurations should be done to the application server (servlet containers) where a signature validation service and policy administration service are deployed.

The configuration steps are described for deployment on an Apache Tomcat server.

2.1.1.1 Adjusting the heap memory

Heap memory need to be set to suitable size that allows normal operations. The required amount of heap memory may vary according to the workload of the server, and the size of documents it is handling. The following settings is an example on how to set heap memory.

Create new file: /%CATALINA_HOME%/bin/setenv.sh (Unix) or setenv.bat (windows) or amend any existing version of this file with the following statement:

```
CATALINA_OPTS="-Xms512m –Xmx2048m"
```

(512mb committed memory and 2048mb max heap memory)

2.1.1.2 Perm Gen memory

The Permanent Generation memory (PermGen) holds metadata about all java classes. It is normally not problem to run the Signature support service in Tomcat using standard PermGen memory settings. However, if have other applications running on the same server, then adding sign support may cause you to run out of PermGen memory. In such case, the PermGen memory space needs to be increased. This is done by adding the following option to CATALINA_OPTS in the setenv.sh file described in the previous section:

```
–XX:MaxPermSize=256m
```

The whole setting could look like:

```
CATALINA_OPTS="-Xms512m –Xmx2048m" –XX:MaxPermSize=256m
```

Note: the values above are only examples. You need to determine what memory size that is appropriate for your environment.

2.1.1.3 Adding class path to the external libraries:

The sign support service makes use of libraries that may not not packaged with the .war deployment builds as doing so may causes problems with Tomcat class loading.

The following library must be provided outside of the .war:

- The IAIK Crypto Library (**iaik_jce_full.jar)**

This library needs to be added to the Tomcat class loader separately, which may be accomplished through the following steps:

1) Edit the file: /%CATALINA_HOME%/conf/catalina.properties

Add the following information to the “common loader” settings:

```
%LOCATION_OF_LIBS%/*.jar
```

where %LOCATION_OF_LIBS% is the location of added external library .jar files (e.g. “/Library/TomcatLibs”).

Example:

```
common.loader=${catalina.base}/lib,${catalina.base}/lib/*.jar,${catalina.home}/lib,${catalina.home}/lib/*.jar,/Library/TomcatLibs/*.jar
```

2) Place the referenced jar files in the specified location (e.g. in “/Library/TomcatLibs/” following the example above).

Note: The IAIK library is a licensed product that can be freely used for evaluation and test purposes. A future release of the support web service can be made without using this library.

The IAIK library needed for the current release of this support service is located in the bundle under the folder “Libs”

3 Configuration files

Configuration and data files are located under the home directory for data storage.

The directory for data storage is set in the web.xml deployment descriptor file for the deployed service under the context-param “DataDir”:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <description>Service Provider support service. This service generates signing requests and handles signing responses from a central signing service</description>
  <display-name>Central Signing SP Support service</display-name>
  <context-param>
    <description>Storage data directory</description>
    <param-name>DataDir</param-name>
    <param-value>/Library/Application Support/EidSigServer/SpSupport</param-value>
  </context-param>
  <context-param>
    <description>Max sign session length in minutes</description>
    <param-name>SignSessionMaxAge</param-name>
    <param-value>10</param-value>
  </context-param>
  <listener>
    <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>SpSupportServlet</servlet-name>
```

```
        <servlet-class>com.aaasec.sigserv.csspsupport.SpSupportServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>SpSupportWs</servlet-name>
        <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpSupportServlet</servlet-name>
        <url-pattern>/spsupport</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>SpSupportWs</servlet-name>
        <url-pattern>/SpSupportWs</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

After deploying this service, set the DataDir (if necessary) to a suitable directory of your service and make sure that the deployed service has read and write access to this directory. Default configuration files (described in the following subsections) and a service certificate will be created automatically in this directory when the SpSupportServlet of this webb application is called for the first time.

The context-param "SignSessionMaxAge" can usually be left unchanged. This is the max age before data related to a particular sign request is deleted from the support service. The value is specified in minutes.

3.1 General configuration (spSupportConfig.json)

This config file specifies a number of important parameters for the support service:

```
{
 "spEntityId": "https://eid2cssp.3xasecurity.com/sign",
 "spServiceReturnUrl": "  https://eid2cssp.3xasecurity.com/sign/SpServlet ",
 "sigServiceEntityId": "https://eid2csig.konki.se/sign",
 "sigServiceRequestUrl": "/CsSigServer/SigRequest",
 "validationServiceUrl": "https://tsltrust.3xasecurity.com/sigval/TTSigValServlet",
 "certType": "QC/SSCD",
 "sigAlgo": "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256",
 "loa": "http://id.elegnamnden.se/loa/1.0/loa3"
}
```

Parameter	Value
spEntityId	The SAML federation EntityID of the service provider that requests signing by the signature service. This is a default value that can be overruled by values supplied through the HTTP or Java API.
spServiceReturnUrl	The return URL where the signing service should send back the sign response message. This is a default value that can be overruled by values supplied through the HTTP or Java API.
sigServiceEntityId	The SAML federation EntityID of the central signing service
sigServiceRequestUrl	The URL where requests to the central signing service should be sent:
validationServiceUrl	A URL to a TSL Trust validation service capable of validating the signed document and returning a validation report. Such service is available from "https://tsltrust.3xasecurity.com/sigval/TTSigValServlet". This service can validate signatures produced by the test signature service when used with the validation policy "All EU Trust Services".
certType	The type of the requested certificate, having one of the following supported values: • "QC" for a Qualified Certificate • "QC/SSCD" for a Qualified Certificate where the private key is declared to be protected by a Secure Signature Creation Device. • "PKC" for a regular Public Key Certificate (which is not declared to be a Qualified Certificate.
sigAlgo	The default requested signature algorithm for requested signatures represented by an URI identifier.
loa	The requested Level Of Assurance with which the signer must be identified when signing. This is an URI identifier of the Authentication Context Class that represents the requested level of assurance in the SAML assertion used to authenticate the signer.

3.2 Attribute map configuration (attrMap.json)

This configuration file allows specification of which attributes that the requesting service prefers/requires in the signer's identity of the generated signer certificate:

```
{
 "attrMap": {
  "rdn:2.5.4.5": {
   "required": true,
   "friendlyName": "serialNumber",
   "samlAttributeNames": [
    {
     "name": "urn:oid:1.2.752.29.4.13",
     "order": 0
    },
    {
     "name": "urn:oid:0.9.2342.19200300.100.1.3",
     "order": 1
    }
   ]
  },
  "rdn:2.5.4.6": {
   "required": true,
   "friendlyName": "country",
   "defaultValue": "SE",
   "samlAttributeNames": [
    {
     "name": "urn:oid:2.5.4.6",
     "order": 0
    }
   ]
  },
  "rdn:2.5.4.42": {
   "required": true,
   "friendlyName": "givenName",
   "samlAttributeNames": [
    {
     "name": "urn:oid:2.5.4.42",
     "order": 0
    }
   ]
  },
  "san:1": {
   "required": false,
   "friendlyName": "e-mail",
   "samlAttributeNames": [
    {
     "name": "urn:oid:0.9.2342.19200300.100.1.3",
     "order": 0
    }
   ]
  },
  "rdn:2.5.4.3": {
    "required": false,
    "friendlyName": "commonName",
    "samlAttributeNames": [
      {
        "name": "urn:oid:2.16.840.1.113730.3.1.241",
        "order": 0
      },
      {
        "name": "urn:oid:2.5.4.3",
        "order": 1
      }
    ]
  },
  "rdn:2.5.4.4": {
    "required": true,
    "friendlyName": "surname",
    "samlAttributeNames": [
      {
        "name": "urn:oid:2.5.4.4",
        "order": 0
      }
    ]
  }
 }
}
```

A certificate name type identifier representing a certificate name type, which is requested to be present in the signer certificate, specifies each object in the attrMap.

The certificate name type identifier is a colon separated string holding two parameters, where the first parameter is one of the values "rdn","san" or "sda". Their meaning and it's associated second parameter is given by the following table:

JSON object	Meaning	Second parameter
rdn	The certificate name type is a relative distinguished name attribute stored in the subject field	An object identifier (OID) identifying the requested attribute (e.g. "rdn:2.5.4.4")
san	A subject alternative name stored in the subject alternative name extension	If the subject alternative name is of type GeneralName, then this parameter is the explicit tag index for the name type. The only supported index is

		1, representing an e-mail address, e.g “san:1”. If the subject alternative name is an OtherName type, then this parameter is an object identifier (OID) identifying the other name type.
sda	A subject directory attribute placed in the subject directory attributes extension.	An object identifier (OID) identifying the attribute stored in the subject directory attributes extension.

The value of each certificate name type object is a sequence of the following JSON objects:

JSON object	Value
required	This parameter is set to **`true`** if the attribute must be present in the signer certificate, otherwise **`false`** (Note that the value is of type Boolean and thus provided without quote characters).
friendlyName	A friendly name of the attribute to be used for UI purposes.
defaultValue	A default value suggested by the requesting service. Whether this suggested default value is accepted or not by the signing service is a matter of policy at the discretion of the signing service.
samlAttributeNames	An array of SAML attribute names that is supposed to be the source of the attribute value (as it is imported from the SAML assertion provided for the user when the user authenticates to the signature service). Each SAML attribute names are specified through a “name” and “order” JSON object.
name	A string representation of the SAML attribute name expressed as a URI. Object identifiers are specified as a URN, having the Name Space Identifier “oid”.
order	An integer representing the preferred order of this SAML attribute as value source for the requested certificate attribute. Attributes with lower order integer values have priority over attributes with higher order

	values.

3.3 Certificate for export

The signature certificate of the support service is located in the same “conf” folder as the configuration files above. The certificate is provided in the file “spCert.crt”.

4 Using the Web Services API

The sign support Web Service operations are described in a WSDL file available from:

{path to service}/SpSupportWs?wsdl

When the provided .war is deployed on a Tomcat server, it will typically be available from the local server at:

http://localhost:8080/CsSpSupport/SpSupportWs?wsdl

The Web service API and its operations are described in a separate API specification.

5 Installing and using the Java API

The servlet provided by the deployed web service receives requests and return responses using standard HTTP GET and POST requests.

This API was the original API that was provided with this support service. This API is less functional than the fully developed WS API and this API may be removed in future releases of this sign support service. It is therefore advisable to use the WS API instead.

IN this Java API, requests and responses provided by this service allows a consuming service to submit a document for signing and receiving a signrequest prepared for the signing service. In addition it allows submission of a corresponding sign response and receiving back a signed version of the submitted document.

This java API is provided by the java class “SignSupportAPI.java”.

In addition to this two additional data java classes are provided to support conversion between JSON and Java objects:

- ServiceStatus.java
- IdAttribute.java

These classes are aimed to be used together with the gson library from google.code:

```
  <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.1</version>
  </dependency>
```

Having the 2 provided data classes in your code, allows you to convert the received status message received upon submitting a signResponse message, into a java object, by using gson to convert the JSON string to a ServiceStatus object.

The Java API is documented using JavaDoc in the provided bundle.

6 Example implementation of the APIs

An example implementation of the WS and Java API is provided in the bundle as a simple MAVEN project in the “SignDemo” folder.