Project information

Customer/Project

OSIF

Document type

API Specification

Title

OSIF 2.1 for CGI eID service

Doc. ID	Version
OSIF 2_1 Eng eID service.doc	1.9
Produced by	**Saved**
CGI	2013-02-18
Approved by	**Date** **Signature**
funktionstjanster@logica.com	

Contents

1 Introduction

This document is the API description of the Nexus MultiID Server (NMS) implementation of the OSIF interface. The NMS implementation is based on OSIF 2.1.

CGI eID Service

In separate frames in this document, there are specific information and comments regarding the OSIF implementation in CGIs eID service. This concerns the normal customers connections to the eID service and use of the Swedish certificates. "Svensk e-legitimation.

There is a test eID environment and a production eID environment. How to communicate with these eID systems are described in the document "Teknisk Bilaga".

We also recommend you to read:

- "Handledning och exempel. Anslutning till eID tjänsten" if you can read Swedish, otherwise read "Information in English"
- "Nexus MultiID Server 3.X Developer´s Guide" chapter *Integration* and chapter *Webservice Interface* and *Demo Application*.
- "Nexus Personal Technical Description" for *WebSigner-*, *Signer2-*, *Authentication-*, and *Version Plug-in*
- "Net iD Developer´s Guide" for the *Sign* plug-in

Test certificates, documentations, recommendations, guidelines, pki-klients/*provider*, example code, CA-certificates etc you will find in CGIs eID Portal.

https://funktionstjanster.primeportal.com/eID

Support and user accounts to eID portal ore ServiceID, pleas contact us at

funktionstjanster@logica.com

OSIF (Offentlig sammanhållen identifieringsfunktion), is a WS/SOAP API for signature validation defined by SVEID (Samverkansgruppen för Elektronisk IDentifikation). Today the Swedish agency Kammarkollegiet (Legal, Financial and Administrative Service Agency) own the OSIF specification.

Generating code from WSDL in Java:
<java-sdk-home>/bin/wsimport http://eidt.funktionstjanster.se:18899/osif/?wsdl

.NET

<net-sdk-home>/bin/wsdl http://eidt.funktionstjanster.se:18899/osif/?wsdl

2 Data types

This chapter describes some common compound data types used in the requests and responses.
The input and output data to the OSIF server, is based on the use of ”*complex types*”, i.e. a data type that consists of a number of subordinated data types. For C programmers this corresponds to a *struct*, and for VB programmers it corresponds to a T*ype*. In this document however, we use the programming language neutral concept of *complex type*.

In all Request-types the following three common parameters exists:

Name	Data type	Occurs	Description	Optional
provider	xsd:int	[1..1]	Type of client, see Provider on page 18.	
transactionID	xsd:string	[0..1]	A transaction ID in the request. Used for tracking purposes. The transactionID that is sent in the request will be returned in the response. If no transactionID was given in the request, the server will generate one and return it in the response.	X
policy	xsd:string	[1..1]	Customer specific ServiceID.	

CGI eID service

provider
Three different providers are used in Sweden, normally you let the user choose between Nordea, Telia or BankID.
*4 **Nexus Personal** for Steria e-tjänstelegitimationer*
5 **Net iD** for Telia
6 BankID Säkerhetsprogram for BankID (**BISP**) for BankID and Nordea
(for other provider id, please contact CGI for more information)

Note:
Provider 4 use ssl authentication and WebSigner Plug-in and was used by Nordea, but now you shall use *Provider* 6 for Nordea. (*Provider 6* use Authentication Plug-in and Signer2 Plug-in)

CGI also support other *provider* ID for special configurations.

transactionID
We recommend you to use transactionID. It will be helpful if tracing transaction between your application and eID service.

policy

Every customer in the eID-service receives a unique “ServiceID” that must be used in the policy. You will receive different ServiceID to be used in test and production eID-service environment. You find your ServiceID in “Teknisk Bilaga”

2.1 Status

```
Begin
      Integer    errorGroup
      String     errorGroupDescription
      Integer    errorCode
      String     errorCodeDescription
End
```

Description

Status contains codes and descriptions that are returned by all operations.

Schema

```
<complexType name="Status">
 <sequence>
  <element name="errorGroup" type="xsd:int"/>
  <element name="errorGroupDescription" minOccurs="0" type="xsd:string"/>
  <element name="errorCode" type="xsd:int"/>
  <element name="errorCodeDescription" minOccurs="0" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Description
errorGroup	xsd:int	[1..1]	Error group code.
errorGroupDescription	xsd:string	[0..1] [1]	Error group description.
errorCode	xsd:int	[0..1]	Error code.
errorCodeDescription	xsd:string	[1..1] [1]	Error code description.

The value for errorGroup can be any of the following with corresponding suggested error text:

errorGroup	errorDescription
1	Your digital id (certificate) has expired and needs to be renewed. Contact your Internet bank in order to retrieve a new.
2	Your digital id (certificate) is revoked and cannot be used. Contact your Internet bank in order to retrieve a new.
3	Your digital id is incorrect. Contact your Internet bank to retrieve a new.
4	Your digital id (certificate) will expire soon. Contact your Internet bank to retrieve a new, OR your digital id will expire within X

[1] An operation completed without errors generates 0 (zero) in *errorGroup* and *errorCode*, while *errorGroupDescription* and *errorCodeDescription* are omitted, or alternatively returned as empty strings

	days. Contact your Internet bank in order to retrieve a new.
5	Technical error. Please try later.
6	Service currently not available. Please try later.

The specific values possible for errorCode and errorCodeDescription is not standardized here and may differ depending on implementation.

Note that the only common value for errorCode and errorCodeDescription among different implementations is the case 0 and blank, i.e. OK. The other values are to be considered as supplier specific.

CGI eID Service

errorGroupDescription and *errorCodeDescription*:. We use Swedish text and you can find them in the document \Generellt\diverse\felkoder\Felkoder NexusMultiID Server.doc together with further explanation of every error code. CGI can change these texts if necessary globally, so if you want to present your own messages to the user, then map them on errorGroup or errorCode.

We recommend that you present the text in *errorGroupDescription* as error message to your customer.

2.2 Property

```
Begin
      String name
      String value
End
```

Description

Property is a compound type that consists of a name and its associated value. Values for name are defined in Attributes on page 20.

Schema

```
<complexType name="Property">
 <sequence>
  <element name="name" type="xsd:string"/>
  <element name="value" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	*Occurs*	Description
name	xsd:string	*[1..1]*	The name of the property.

value	xsd:string	*[1..1]*	The value of the property.

2.3 GenerateChallengeRequest

CGI eID Service
GeneratChallenge are only used in authentication with BankID/Nordea (*provider*=6). An application level authentication. Read *Nexus MultiID Server 3.0 Developers Guide* for more information and the flow of application level authentication. Also read *Nexus Personal Technical Description – Authentication Plug-in* You can also use *generateChallenge*, and use the *challenge* if you need a random base 64-encoded number for provider = 6. (Hex-string-encoden for provider 3)-For example you need that for "Nonce" value in Signing process with *provider 6* (Signer2 Plug-in)

```
Begin
      Integer provider
      String transactionID
      String policy
End
```

Description

Input data for the GenerateChallenge operation.

Schema

```
<complexType name="GenerateChallengeRequest">
 <sequence>
  <element name="provider" type="xsd:int"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="policy" minOccurs="0" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment	Optional
provider	xsd:int	[1..1]	Type of client. BankID/Nordea = 6	
transactionID	xsd:string	[0..1]	An ID for the transaction.	X
policy	xsd:string	[1..1]	Your customer specific ServiceID	

2.4 GenerateChallengeResponse

```
Begin
      Status status
      String challenge
End
```

Description

GenerateChallengeReponse contains statuses and generated results[2].

Schema

```
<complexType name="GenerateChallengeResponse">
 <sequence>
  <element name="status" type="impl:Status"/>
  <element name="challenge" minOccurs="0" type="xsd:string"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment
status	*impl:status*	*[1..1]*	*Status for the operation.*
challenge	*xsd:string*	*[0..1]*	A challenge sent to the client, in order to be signed, for login purpose. The value should also be saved in the server session and used as the input data for the *VerifyAuthentication* operation. Depending on the client, the challenge is encoded in the following way (indexed per provider). *Only used by BankID. The challenge is Base64 encoded.*
transactionID	*xsd:string*	*[0..1]*	An ID for the transaction.

2.5 EncodeTBSRequest

CGI eID Service
EncodeTBS help you to prepare the text that should be signed by the client. *TBS (*To Be Signed). Info:- for provider 5 (Telia) TBS will be URL-encoded UTF-8 and for provider 6 (BankID/Nordea) TBS will be Base-64 encoded UTF-8. *EncodeTBS* are used in signing in an application level by all *provider*

```
Begin
      Integer provider
```

[2] Common for all Response-types is that if the operation reports error, then all fields in Response-type will be considered optional (except status).

```
      String transactionID
      String policy
      String tbsText
End
```

Description

Input data to EncodeTBS-operation.

Schema

```
<complexType name="EncodeTBSRequest">
 <sequence>
  <element name="provider" type="xsd:int"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="policy" minOccurs="0" type="xsd:string"/>
  <element name="tbsText" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment	Optional
provider	xsd:int	[1..1]	Type of client.	
transactionID	xsd:string	[0..1]	An ID for the transaction.	X
policy	xsd:string	[1..1]	Your customer specific ServiceID	
tbsText	xsd:string	[1..1]	Text to be signed.	

2.6 EncodeTBSResponse

```
Begin
      Status status
      String text
End
```

Description

EncodeTBSResponse contains status and encoded text. Output from *EncodeTBS.*

Schema

```
<complexType name="EncodeTBSResponse">
 <sequence>
  <element name="status" type="impl:Status"/>
  <element name="text" minOccurs="0" type="xsd:string"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment
status	impl:status	[1..1]	*Status for the operation.*
text	xsd:string	[0..1]	A text to be sent to the client. The value should also be saved in the server session and be given as input

			data to then *VerifySignature* operation. *Depending on the type of client the conversion is performed in the following way:* [provider = 6 *BankID/Nordea, BISP*] Base64-encoded. [provider=5 *Telia, NetiD*] HTML-encoded.
transactionID	xsd:string	[0..1]	An ID for the transaction.

2.7 VerifyCertificateRequest

CGI eID Service
VerifyCertificate are only used by provider 5 (Telia) in ssl-based authentications. All CA-certificate must be installed in the Web Servers "trusted store" and the application must receive the client certificate from the Web Server. You find all CA-certificate for test and production in CGIs eID Portal.

```
Begin
      Integer provider
      String transactionID
      String policy
      String array certificates
      Integer checkUsage
End
```

Description

Input data entry to the VerifyCertificate-operation.

Schema

```
<complexType name="VerifyCertificateRequest">
 <sequence>
  <element name="provider" type="xsd:int"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="policy" minOccurs="0" type="xsd:string"/>
  <element name="certificates" minOccurs="1" maxOccurs="unbounded"
type="xsd:string"/>
  <element name="checkUsage" type="xsd:int"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment	Optional
provider	xsd:int	[1..1]	Type of client.	
transactionID	xsd:string	[0..1]	An ID for the transaction.	**X**

policy	xsd:string	[1..1]	Your customer specific ServiceID	
certificates	xsd:string	[1..*]	One or more certificates in an array that should be sorted with the end-user certificate as element 0. The certificates shall be Base64 encoded , X509v3.	
checkUsage	xsd:int	[1..1]	States possible control of the certificate's keyUsage, see CheckUsage on page 22.	

CGI eID Service
checkUsage You will for a normal implementation only use "-1", then we make the keyUsage check according to the rules of each Provider.

2.8 VerifyCertficateResponse

```
Begin
      Status status
      Property Array attributes
      String userID
      String transactionID
End
```

Description

VerifyCerticateResponse contains status, attributes, identity on members and identity of the transaction. Output from *VerifyCertificateRequest.*

Schema

```
<complexType name="VerifyCertificateResponse">
 <sequence>
  <element name="status" type="impl:Status"/>
  <element name="attributes" minOccurs="0" maxOccurs="unbounded"
type="impl:Property"/>
  <element name="userID" minOccurs="0" type="xsd:string"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="certificate" minOccurs="0" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment	Optional
status	impl:status	[1..1]	Status for the operation.	
attributes	xsd:string	[1..*]	One or several certificate attributes, see Attributes on page 20.	
userID	xsd:string	[0..1]	The user certificate	

transactionID	xsd:string	[0..1]	Identity of the transaction.	X
certificate	xsd:string	[0..1]	The user certificate: Base64-encoded certificate X509 v3.	

CGI eID Service

userID and certificate give the same client certificate in Base64 encoded, 509v3. userID could be used for other things in the future, so don't use that.

The use of certificate/*userID* are a bit of obsolete when BankID no longer have that requirement that the same certificate used in authentication, must later on be used in a process if a signing is required.

2.9 VerifyAuthenticationRequest

CGI eID Service

VerifyAuthentication are normally only used in authentication with BankID/Nordea (*provider*=6) after *GenerateChallenge* and you received GetParam "signature" from Nexus Personal Authentication Plug-in.

```
Begin
      Integer provider
      String transactionID
      String policy
      String challenge
      String signature
      String host
      String hiddenTbs
End
```

Description

Input data to the VerifyAuthentication-operation.

Schema

```
<complexType name="VerifyAuthenticationRequest">
 <sequence>
  <element name="provider" type="xsd:int"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="policy" minOccurs="0" type="xsd:string"/>
  <element name="challenge" type="xsd:string"/>
  <element name="signature" type="xsd:string"/>
  <element name="host" minOccurs="0" type="xsd:string"/>
  <element name="hiddenTbs" minOccurs="0" type="xsd:string"/>
</sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment	Optional
provider	xsd:int	[1..1]	Type of client.	
transactionID	xsd:string	[0..1]	An ID for the transaction.	X

policy	xsd:string	[1..1]	Your customer specific ServiceID	
challenge	xsd:string	[1..1]	The challenge signed by the client at log-in. This parameter shall come from the server session and corresponds to the *challenge*-parameter in *GenerateChallengeResponse*.	
signature	xsd:string	[1..1]	The message returned from the client. It shall be encoded as it normally comes from the client: [provider=6 *BankID/Nordea, BISP*] Unencoded (string). Provider=5 [*Telia, NetiD*] Base64-encoded.	
host	xsd:string	[0..1]	Name of external web server (FQDN).	X
hiddenTBS	xsd:string	[0..1]	Some clients support non visible data to be signed. In case that is used, this parameter should originate from the server session.	X

CGI eID Service

host – you don't have to use this parameter. (obsolete).

hiddenTbs – NonVisibleData that can be signed, Function not supported by any provider today. Don't use.

Ensure that the *challenge* stored on the server side (application) are stored for every user session and not as a global parameter.

2.10 VerifyAuthenticationResponse

```
Begin
      Status status
      String Array attributes
      String userID
      String transactionID
      String certificate
End
```

Description

VerifyAuthenticationResponse contains status, certificate attributes, the id of the user and the user certificate.

Schema

```
<complexType name="VerifyAuthenticationResponse">
 <sequence>
  <element name="status" type="impl:Status"/>
  <element name="attributes" minOccurs="0" maxOccurs="unbounded"
type="impl:Property"/>
  <element name="userID" minOccurs="0" type="xsd:string"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="certificate" minOccurs="0" type="xsd:string"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment
status	impl:status	[1..1]	Status for the operation.
attributes	xsd:string	[0..*]	Zero, one or more certificate attributes. see Attributes on page 20
userID	xsd:string	[0..1]	The user certificate
transactionID	xsd:string	[0..1]	Identity of the transaction.
certificate	xsd:string	[0..1]	*The (base64-encoded X509 v3) user certificate used to sign the challenge*

CGI eID Service
userID The user certificate base64 encoded. Can be used in VerifySignatureRequest to make sure that the same certificate/user that made an authentication is the same certificate/user later making a signature. If you build in this logic in your application you don't allow the user to use another certificate in the signing process. If you after an authentication want to make sure that the same individual later on perform a signing, we recommend you to use the different filter functionality at each signer Plug-in to only allow certificate with a specific Subject.SerialNumber to be used.

2.11 VerifySignatureRequest

CGI eID Service
VerifySignature are used by all *providers* in the signing process

```
Begin
      Integer provider
      String transactionID
      String policy
      String tbsText
```

```
      String userID
      String signature
      String host
      String hiddenTbs
      String nonce
      String file
End
```

Description

Input data to VerifySignature-operation.

Schema

```
<complexType name="VerifySignatureRequest">
 <sequence>
  <element name="provider" type="xsd:int"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="policy" minOccurs="0" type="xsd:string"/>
  <element name="tbsText" type="xsd:string"/>
  <element name="userID" minOccurs="0" type="xsd:string"/>
  <element name="signature" type="xsd:string"/>
  <element name="host" minOccurs="0" type="xsd:string"/>
  <element name="hiddenTbs" minOccurs="0" type="xsd:string"/>
  <element name="nonce" minOccurs="0" type="xsd:string"/>
  <element name="file" maxOccurs="1" minOccurs="0"
type="Base64Binary"/>
 </sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment	Optional
provider	xsd:int	[1..1]	Type of client.	
transactionID	xsd:string	[0..1]	An ID for the transaction.	X
policy	xsd:string	[1..1]	Your customer specific ServiceID	
tbsText	xsd:string	[1..1]	Text to be signed. This parameter should come from the server session, and corresponds to the *text* parameter in *EncodeTBSResponse*.	
userID	xsd:string	[0..1] [3]	The value should originate from the server session and corresponds to the userID in VerifyAuthenticationResponse. If this value is included, the server may check to see that the authentication and signing certificate corresponds.	X
signature	xsd:string	[1..1]	Message as returned from the client. This should be coded in the way that it normally comes from the client*:* [provider=6 *BankID/Nordea*,	

[3] Strictly speaking, this and the previous parameter are optional *in certain cases*, but we propose that they are treated as compulsory to simplify implementation!

			BISP] Unencoded (string). [provider=5 *Telia*, *NetiD*] Base64-encoded.	
host	xsd:string	[0..1]	Name of external web server (FQDN).	X
hiddenTBS	xsd:string	[0..1]	Some clients support non visible data to be signed. In case that is used, this parameter should originate from the server session.	X (can be used for provider 6)
nonce	xsd:string	[0..1]	Random value to complicate replaying attacks. The value is generated through a call to the operation *GenerateChallenge*. This parameter is mandatory for BankID. this parameter sholud originate from the server side.	(X) (mandatory for provider 6)
file	Base64Binary	[0..1]	För BankID and file signing function with .pdf or .txt files. (Extension to OSIF 2.1)	(X) (only for file-signing function in BankID)

CGI eID Service

Nonce – this is required by provider=6 (BankID/Nordea) and its purpose is to obstruct abuse based on retransmitting transactions. You can cerate an own random number (base64 encoded) ore use a returned transactionID or Challenge?

Ensure that the *tbsText* (*EncodeTBSResponce – text*) stored on the server side (application) are stored for every user session and not as a global parameter.

We recommend you not to use, *userID, hiddenTBS, or host, if you don't need to.*

File – Is only used with BankID special file signing function. (BISP Signer2 and FileContent)

2.12 VerifySignatureResponse

```
Begin
      status status
      Property Array attributes
      String transactionID
      String certificate
End
```

Description

VerifySignatureResponse contains status, attributes, identity on the transaction and certificates.

Schema

```
<complexType name="VerifySignatureResponse">
 <sequence>
  <element name="status" type="impl:Status"/>
  <element name="attributes" minOccurs="0" maxOccurs="unbounded"
type="impl:Property"/>
  <element name="transactionID" minOccurs="0" type="xsd:string"/>
  <element name="certificate" minOccurs="0" type="xsd:string"/>
</sequence>
</complexType>
```

Element description

Name	Data type	Occurs	Comment
status	impl:status	[1..1]	Status for the operation.
attributes	xsd:string	[0..*]	Zero, one or more certificate attributes. see Attributes on page 20.
transactionID	xsd:string	[0..1]	*An ID for the transaction.*
certificate	xsd:string	[0..1]	Base64-encoded certificate X509 v3.

3 Constant values

3.1 Provider

The attribute is used to be able to identify the PKI client (issuer) in order to meet demands from different certificate issuers and corresponding PKI clients.

ID	Organisation	Pki-klient	Comment
1	BankID - old	In OSIF 2.0 BankID CBT	Can be used instead of provider 6, from OSIF 2.1
2	Nordea - old	In OSIF 2.0 SmartTrust Personal	Can be used instead of provider 4, from OSIF 2.1
3	Telia - old	In OSIF 2.0 VeriSign PTA	Can be used instead of provider 5, from OSIF 2.1
4	Nordea – old but allowed to use until 2014.	Nexus Personal	Use ssl for identification and WebSigner plug-in for signing.
5	Telia, SITHS, SignGuard.	Netmaker NetID	Use ssl for identification and iid plug-in for signing.
6	BankID and Nordea	BISP (BankID Säkerhetsprogram)	use Autentification plug-in for identification and Signer2 plug-in for signing.
7	BankID i mobil (BIM) - old	WPKI	A new BMI:api exist för Mobilt BankID
8-19			Reserved by Kammarkollegiet

CGI eID Service	
The provider that are used in Sweden today is:	
PKI-klient	CA
5 – Net iD	Telia, (SITH, Steria, SignGuard)
6 – Personal 4.18 (BISP)	BankID and Nordea

Special provider: (pleas consult with CGI before use)
33 for “challenge” based authentication with Telia and NetiD

More information about each (pki-klient) *provider* do you find in CGIs eID Portal.

3.2 Attributes

Names that are used in attributes.

Name	Description	Comment
SerialNumber	Client certificate serial number	
Issuer.CommonName	Name of CA	
Issuer.OrganizationName	Organization name of CA	
Issuer.CountryName	Country for CA	
Subject.CommonName	Name (given name and surname)	
Subject.GivenName	All given names	
Subject.Surname	Surname	
Subject.SerialNumber	Swedish social security number (personnummer)	
Subject.CountryName	Country	
Expiredate[4]	Date when the certificate expires	
NotBeforeDate	Creation time for the certificate	
securitylevel.level	level 1-4	
securitylevel.description	Eg. SoftwarePKI, SmartcardPKI, MobileTwofactorContract (wpki)	
validation.type	Revocation by OSIF or CRL	
validation.response	OCSP-receipt or information about the CRL that been used.	
validation.ocsp.response	Do not use. Will be removed.	
usercert.subject.title		*If included e-tjänsteleg*
usercert.subject.emailAddress	*e-mail adress*	*If included e-tjänsteleg*
usercert.subject.localityName	*Geographic location.*	*If included e-tjänsteleg*
Subject.OrganizationName	*Name of the organisation*	*e-tjänsteleg*

[4] The date format is YYYYMMDDHHMMSSZ, ex: 20040616135825Z. Z states that the time to be stated in Greenwich Mean Time (GMT).

CGI eID Service

For "Svensk e-legitimation" are the following attributes normal of interest.

Attribut	*Description*	*Example*
Subject.CommoneName (CN)	Name	Agda Andersson
Subject.GivenName (G)	All given names	Agda
Subject.Surname (SN)	Surename	Andersson
Subject.SerialNumber	"personnummer"	188803099368
Subject.ContryName (C)	country	SE

Securitylevel information can be used if you want to restrict access rights based on authentication level. We use level 3 and 4. Level 3 – for all soft certificates and level 4 for certificate on pki-cards.

Validation.response. That should be saved. It is a base64 data receipt including time-stamp, hash value of document, nonce, status and are signed by OCSP responder if validation.type=OCSP. Other vise there are information about the CRL that been used. This can be used as legal evidence. (about 1kb Base64 encoded data)

In Sweden there exist personal organization certificates for employees that can be used in some e-services. (Steria and SITHS).

3.3 CheckUsage

CheckUsage is used in order to indicate a possible check of keyUsage in the certificate.

Value	value
0	No control is carried out
1	Digital Signature
2	NonRepudiation
4	KeyEncipherment
8	DataEncipherment
16	KeyAgreement
32	KeyCertSign
64	CRLSign
128	EncipherOnly
256	DecipherOnly
-1	(Provider specific). This is what you normally will use.

CGI eID Service
checkUsage You will only use "-1", then we make the keyUsage check according to the rules of the Provider.

4 Operations

4.1 GenerateChallenge (only used for BankID)

Syntax

```
GenerateChallengeResponse
GenerateChallenge(GenerateChallengeRequest request)
```

Description

Generates a challenge (random number) that is used at authentication and encodes it in accordance with client requirements.

Schema

```
<wsdl:operation name="GenerateChallenge">
   <wsdlsoap:operation soapAction="GenerateChallenge"/>
   <wsdl:input name="GenerateChallengeInput">
      <wsdlsoap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="GenerateChallengeOutput">
      <wsdlsoap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
```

Parameters

Name	Description
request	Input data. See documentation for data types

Returns

Compound data type that contains information about the operation. See documentation for data type *GenerateChallengeResponse*

4.2 EncodeTBS

Syntax

```
EncodeTBSResponse
EncodeTBS(EncodeTBSRequest request)
```

Description

Convert text for the signature component in accordance with client requirements.

Schema

```
<wsdl:operation name="EncodeTBS">
   <wsdlsoap:operation soapAction="EncodeTBS"/>
   <wsdl:input name="EncodeTBSInput">
     <wsdlsoap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="EncodeTBSOutput">
     <wsdlsoap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
```

Parameters

Name	Description
Request	Input data. See documentation for data types

Returns

Compound data type that contains information about the operation. See documentation for data type *EncodeTBSResponse*.

4.3 VerifyCertificate

Syntax

```
VerifyCertificateResponse

VerifyCertficate (VerifyCertificateRequest request)
```

Description

Verifies a certificate chain.

Schema

```
<wsdl:operation name="VerifyCertificate">
   <wsdlsoap:operation soapAction="VerifyCertificate"/>
   <wsdl:input name="VerifyCertificateInput">
      <wsdlsoap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="VerifyCertificateOutput">
      <wsdlsoap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
```

Parameters

Name	Description
Request	Input data. See documentation for data types

Returns

Compound data type that contains information about the operation. See documentation for data type *VerifyCertificateResponse.*

4.4 VerifyAuthentication

Syntax

```
VerifyAuthenticationResponse

VerifyAuthentication (VerifyAuthenticationRequest
request)
```

Description

Verifies signature sent from the client upon authentication of user.

Schema

```
<wsdl:operation name="VerifyAuthentication">
   <wsdlsoap:operation soapAction="VerifyAuthentication"/>
   <wsdl:input name="VerifyAuthenticationInput">
```

```
      <wsdlsoap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="VerifyAuthenticationOutput">
      <wsdlsoap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
```

Parameters

Name	Description
Request	Input data. See documentation for data types

Returns

Compound data type that contains information about the operation. See documentation for data type *VerifyAuthenticationResponse*.

4.5 VerifySignature

Syntax

```
VerifySignatureResponse
VerifySignature(VerifySignatureRequest request)
```

Description

Verifies signature sent from the client when signing text.

Schema

```
<wsdl:operation name="VerifySignature">
   <wsdlsoap:operation soapAction="VerifySignature"/>
   <wsdl:input name="VerifySignatureInput">
      <wsdlsoap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="VerifySignatureOutput">
      <wsdlsoap:body use="literal"/>
   </wsdl:output>
</wsdl:operation>
```

Parameters

Name	Description
Request	Input data. See documentation for data types

Returns

Compound data type that contains information about the operation.

See documentation for data type *VerifySignatureResponse*.

5 Error handling

If an operation has completed OK it is indicated by errorGroup and errorCode are set to 0 in the status parameter.

An operation that results in an error is indicated by errorGroup and errorCode is set to indicate error type and error code. This also indicates that all parameters in the response, apart from statuses, become optional. This means that when an error occurs, only the error information is returned.

An application must also handle other types of errors, ex SOAP- fault or HTTP errors if the request was badly formatted.

CGI eID Service
Read more about error and error explanations in the document \Generellt\diverse\felkoder\Felkoder NexusMultiID Server.doc ‘ Always display errorGroupDescription to the user!

5.1 WSDL

5.1.1 Comment

The WSDL description below contains no service tag. The service-tag is instead included by the service supplier and is preferably placed after the binding tag. The service tag contains the address to an instance of an OSIF service. Below is shown an example of a service tag:

```
   <wsdl:service name="OsifService">
      <wsdl:port name="osif" binding="impl:osifSoapBinding">
         <wsdlsoap:address
location="http://localhost:8080/osif"/>
      </wsdl:port>
   </wsdl:service>
```

5.1.2 OSIF API

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="urn:www.sll.se/wsdl/soap/osif"
xmlns:impl="urn:www.sll.se/wsdl/soap/osif"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

 <wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:www.sll.se/wsdl/soap/osif">

  <complexType name="Status">
    <sequence>
     <element name="errorGroup" type="xsd:int"/>
     <element name="errorGroupDescription" minOccurs="0"
type="xsd:string"/>
     <element name="errorCode" type="xsd:int"/>
     <element name="errorCodeDescription" minOccurs="0"
type="xsd:string"/>
    </sequence>
   </complexType>

   <complexType name="Property">
    <sequence>
     <element name="name" type="xsd:string"/>
     <element name="value" type="xsd:string"/>
    </sequence>
   </complexType>

   <complexType name="GenerateChallengeRequest">
    <sequence>
     <element name="provider" type="xsd:int"/>
     <element name="transactionID" minOccurs="0"
type="xsd:string"/>
     <element name="policy" minOccurs="0" type="xsd:string"/>
```

```
    </sequence>
   </complexType>
   <element name="generateChallengeRequest"
type="impl:GenerateChallengeRequest"/>

  <complexType name="GenerateChallengeResponse">
    <sequence>
     <element name="status" type="impl:Status"/>
     <element name="challenge" minOccurs="0"
type="xsd:string"/>
     <element name="transactionID" minOccurs="0"
type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="generateChallengeResponse"
type="impl:GenerateChallengeResponse"/>

   <complexType name="EncodeTBSRequest">
    <sequence>
     <element name="provider" type="xsd:int"/>
     <element name="transactionID" minOccurs="0"
type="xsd:string"/>
     <element name="policy" minOccurs="0" type="xsd:string"/>
     <element name="tbsText" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="encodeTBSRequest"
type="impl:EncodeTBSRequest"/>

   <complexType name="EncodeTBSResponse">
    <sequence>
     <element name="status" type="impl:Status"/>
     <element name="text" minOccurs="0" type="xsd:string"/>
     <element name="transactionID" minOccurs="0"
type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="encodeTBSResponse"
type="impl:EncodeTBSResponse"/>

   <complexType name="VerifyAuthenticationRequest">
    <sequence>
     <element name="provider" type="xsd:int"/>
     <element name="transactionID" minOccurs="0"
type="xsd:string"/>
     <element name="policy" minOccurs="0" type="xsd:string"/>
     <element name="challenge" type="xsd:string"/>
     <element name="signature" type="xsd:string"/>
     <element name="host" minOccurs="0" type="xsd:string"/>
     <element name="hiddenTbs" minOccurs="0"
type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="verifyAuthenticationRequest"
type="impl:VerifyAuthenticationRequest"/>

   <complexType name="VerifyAuthenticationResponse">
    <sequence>
     <element name="status" type="impl:Status"/>
```

```
      <element name="attributes" minOccurs="0"
maxOccurs="unbounded" type="impl:Property"/>
      <element name="userID" minOccurs="0" type="xsd:string"/>
      <element name="transactionID" minOccurs="0"
type="xsd:string"/>
      <element name="certificate" minOccurs="0"
type="xsd:string"/>
     </sequence>
    </complexType>
    <element name="verifyAuthenticationResponse"
type="impl:VerifyAuthenticationResponse"/>

    <complexType name="VerifySignatureRequest">
     <sequence>
      <element name="provider" type="xsd:int"/>
      <element name="transactionID" minOccurs="0"
type="xsd:string"/>
      <element name="policy" minOccurs="0" type="xsd:string"/>
      <element name="tbsText" type="xsd:string"/>
      <element name="userID" minOccurs="0" type="xsd:string"/>
      <element name="signature" type="xsd:string"/>
      <element name="host" minOccurs="0" type="xsd:string"/>
      <element name="hiddenTbs" minOccurs="0"
type="xsd:string"/>
      <element name="nonce" minOccurs="0" type="xsd:string"/>
     </sequence>
    </complexType>
    <element name="verifySignatureRequest"
type="impl:VerifySignatureRequest"/>

    <complexType name="VerifySignatureResponse">
     <sequence>
      <element name="status" type="impl:Status"/>
      <element name="attributes" minOccurs="0"
maxOccurs="unbounded" type="impl:Property"/>
      <element name="transactionID" minOccurs="0"
type="xsd:string"/>
      <element name="certificate" minOccurs="0"
type="xsd:string"/>
     </sequence>
    </complexType>
    <element name="verifySignatureResponse"
type="impl:VerifySignatureResponse"/>

 <complexType name="VerifyCertificateRequest">
     <sequence>
      <element name="provider" type="xsd:int"/>
      <element name="transactionID" minOccurs="0"
type="xsd:string"/>
      <element name="policy" minOccurs="0" type="xsd:string"/>
      <element name="certificates" minOccurs="1"
maxOccurs="unbounded" type="xsd:string"/>
      <element name="checkUsage" type="xsd:int"/>
     </sequence>
    </complexType>
    <element name="verifyCertificateRequest"
type="impl:VerifyCertificateRequest"/>

 <complexType name="VerifyCertificateResponse">
```

```
    <sequence>
     <element name="status" type="impl:Status"/>
     <element name="attributes" minOccurs="0"
maxOccurs="unbounded" type="impl:Property"/>
     <element name="userID" minOccurs="0" type="xsd:string"/>
     <element name="transactionID" minOccurs="0"
type="xsd:string"/>
     <element name="certificate" minOccurs="0"
type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="verifyCertificateResponse"
type="impl:VerifyCertificateResponse"/>

 </schema>
 </wsdl:types>

   <wsdl:message name="GenerateChallengeInputMessage">
      <wsdl:part name="generateChallengeRequest"
element="impl:generateChallengeRequest"/>
   </wsdl:message>

   <wsdl:message name="GenerateChallengeOutputMessage">
      <wsdl:part name="generateChallengeResponse"
element="impl:generateChallengeResponse"/>
   </wsdl:message>

   <wsdl:message name="EncodeTBSInputMessage">
      <wsdl:part name="encodeTBSRequest"
element="impl:encodeTBSRequest"/>
   </wsdl:message>

   <wsdl:message name="EncodeTBSOutputMessage">
      <wsdl:part name="encodeTBSResponse"
element="impl:encodeTBSResponse"/>
   </wsdl:message>

   <wsdl:message name="VerifyAuthenticationInputMessage">
      <wsdl:part name="verifyAuthenticationRequest"
element="impl:verifyAuthenticationRequest"/>
   </wsdl:message>

   <wsdl:message name="VerifyAuthenticationOutputMessage">
      <wsdl:part name="verifyAuthenticationResponse"
element="impl:verifyAuthenticationResponse"/>
   </wsdl:message>

   <wsdl:message name="VerifySignatureInputMessage">
      <wsdl:part name="verifySignatureRequest"
element="impl:verifySignatureRequest"/>
   </wsdl:message>

   <wsdl:message name="VerifySignatureOutputMessage">
      <wsdl:part name="verifySignatureResponse"
element="impl:verifySignatureResponse"/>
   </wsdl:message>

   <wsdl:message name="VerifyCertificateInputMessage">
      <wsdl:part name="verifyCertificateRequest"
```

```
element="impl:verifyCertificateRequest"/>
   </wsdl:message>

   <wsdl:message name="VerifyCertificateOutputMessage">
      <wsdl:part name="verifyCertificateResponse"
element="impl:verifyCertificateResponse"/>
   </wsdl:message>

   <wsdl:portType name="Osif">

      <wsdl:operation name="GenerateChallenge">
         <wsdl:input name="GenerateChallengeInput"
message="impl:GenerateChallengeInputMessage"/>
         <wsdl:output name="GenerateChallengeOutput"
message="impl:GenerateChallengeOutputMessage"/>
      </wsdl:operation>

      <wsdl:operation name="EncodeTBS">
         <wsdl:input name="EncodeTBSInput"
message="impl:EncodeTBSInputMessage"/>
         <wsdl:output name="EncodeTBSOutput"
message="impl:EncodeTBSOutputMessage"/>
      </wsdl:operation>

      <wsdl:operation name="VerifyAuthentication">
         <wsdl:input name="VerifyAuthenticationInput"
message="impl:VerifyAuthenticationInputMessage"/>
         <wsdl:output name="VerifyAuthenticationOutput"
message="impl:VerifyAuthenticationOutputMessage"/>
      </wsdl:operation>

      <wsdl:operation name="VerifySignature">
         <wsdl:input name="VerifySignatureInput"
message="impl:VerifySignatureInputMessage"/>
         <wsdl:output name="VerifySignatureOutput"
message="impl:VerifySignatureOutputMessage"/>
      </wsdl:operation>

      <wsdl:operation name="VerifyCertificate">
         <wsdl:input name="VerifyCertificateInput"
message="impl:VerifyCertificateInputMessage"/>
         <wsdl:output name="VerifyCertificateOutput"
message="impl:VerifyCertificateOutputMessage"/>
      </wsdl:operation>

   </wsdl:portType>

   <wsdl:binding name="osifSoapBinding" type="impl:Osif">
     <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

     <wsdl:operation name="GenerateChallenge">
        <wsdlsoap:operation soapAction="GenerateChallenge"/>
        <wsdl:input name="GenerateChallengeInput">
           <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="GenerateChallengeOutput">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
```

```
      </wsdl:operation>

      <wsdl:operation name="EncodeTBS">
         <wsdlsoap:operation soapAction="EncodeTBS"/>
         <wsdl:input name="EncodeTBSInput">
           <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="EncodeTBSOutput">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="VerifyAuthentication">
         <wsdlsoap:operation
soapAction="VerifyAuthentication"/>
         <wsdl:input name="VerifyAuthenticationInput">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="VerifyAuthenticationOutput">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="VerifySignature">
         <wsdlsoap:operation soapAction="VerifySignature"/>
         <wsdl:input name="VerifySignatureInput">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="VerifySignatureOutput">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="VerifyCertificate">
         <wsdlsoap:operation soapAction="VerifyCertificate"/>
         <wsdl:input name="VerifyCertificateInput">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="VerifyCertificateOutput">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

   </wsdl:binding>

</wsdl:definitions>
```