AS Sertifitseerimiskeskus

DigiDocService specification

Document version:	**2.123**
Last update:	**01.03.2009**
Service version:	**2.3.30**

Table of contents

1 Document Versions

Ver.	Date	Author	Notes
1.103	31.10.05	Urmo Keskel	The first version, this document is based on document “DigiDocService teenuse mudel ja spetsificatioon” created by Veiko Sinivee.
1.104	07.02.06	Urmo Keskel	Added SmartCard signing functions, SessionCode parameter moved from header to body. Removed exampled.
1.105	03.05.06	Urmo Keskel	Changed methods StartSession, MobileSign and PrepareSignature, added parameter signatureProfile. Added methods describing timestamps and certificate revocations lists.
1.120	20.03.07	Urmo Keskel	Major upgrade
1.122	23.04.07	Urmo Keskel	GetMobileCertificate method added, general text corrections
1.123	19.12.08	Ahto Jaago, Urmo Keskel	- Added section „Suggestions and requirements about using the service“ - Added CheckCertificate method description - Added section „Authentication using ID-card“ - Updated descriptions of following methods and datastructure: StartSession, MobileAuthenticate, MobileAuthenticateStatus, AddDataFile, DataFileInfo - Updated section 6.2.1

2 References

[1] RFC3275	(Extensible Markup Language) XML-Signature Syntax and Processing. March 2002.
[2] ETSI TS 101 903	XML Advanced Electronic Signatures (XAdES). February 2002.
[3] DigiDoc Format	DigiDoc Format Specification http://www.sk.ee/files/DigiDoc%201.3%20file%20format.pdf 12.05.2004
[4] SOAP	Simple Object Access Protocol http://www.w3.org/TR/soap/
[5] Time Formats	The W3C note Date and Time Formats *http://www.w3.org/TR/NOTE-datetime*, September 1997
[6] ETSI TS 102 204	Mobile Commerce (M-COMM); Mobile Signature Service; Web Service Interface. V.1.1.4, August 2003.
[7] RFC 3161	Internet X.509 Public Key Infrastructure: Time-Stamp Protocol (TSP), August 2001

3 Terms and Acronyms

Application Provider	A party who provides the end-user with a digital signing, signature verifying service orauthentication service
Control Code	4-digit number used in mobile authentication and mobile signing which is cryptographically linked with hash value to be signed. Control Code is displayed both in mobile phone and computer application in order to provide for authencity of the signing request.
Hash, Hash value	Data to be signed which is cryptographically derived from Datafiles and other parameters to be signed
Mobile-ID	Service based on Wireless PKI providing for mobile authentication and digital signing. Mobile-ID user uses special SIM card with private keys on it. Hash to be signed is sent over the GSM network to the phone and the user shall enter PIN code to sign. The signed result is sent back over the air.
MSSP	Mobile Signature Service Provider. Described in stardard ETSI TS 102 204 [6].
Original file, Datafile	File to be digitally signed. The file is in arbitrary file format
Signing	Used in this case as „forming the digital signature" according to the Digital Signature Law. The procedure includes besides signing the validity confirmation getting.
Verification	Checking the validity of signatures of the digitally signed data.
Transaction, session	Communication while a file (DigiDoc or the original datafile) is forwarded to the webservice and some operations related to these are followed f.e. a DigiDoc is created out of the data file, then signed and returned to the application. After closing the transaction all the information created during the transaction is deleted from

	the service-server

4 Introduction

DigiDoc is a SOAP-based webservice enabling a pretty easy integration for the functionality of digital signing, verifying signatures and authentication with other information systems. The service is usable in different development environments and platforms featuring SOAP 1.0-encoded support.

Functionality of the service:

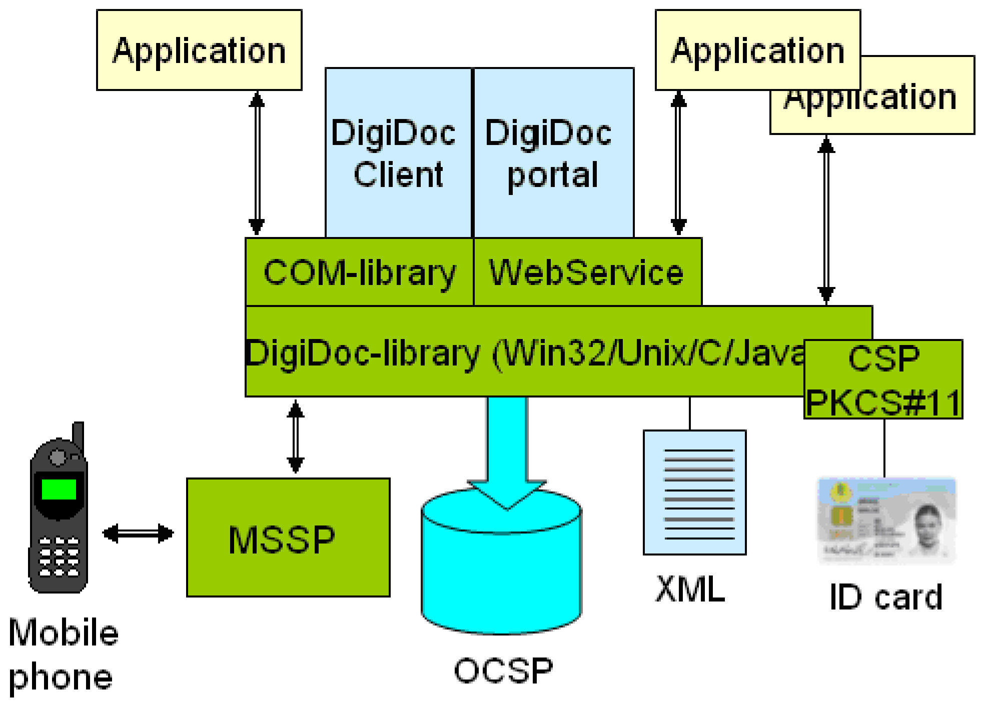
- Authentication with Mobile-ID
- Verification of certificate’s validity (incl smartcard)
- Creation of DigiDoc files
- Digital signing with Mobile-ID
- Digital signing with smartcard
- Verification of digitally signed files and validity of signatures

Access to the service is IP-based and requires written contract with provider of DigiDocService.

4.1 About DigiDoc

DigiDoc is a universal system for creating and verifying digitally signed files according to ETSI TS 101 903 (XAdES) specification. DigiDoc system is also capable of encryption/decryption of files (signed or unsigned) according to W3C XML Encryption Recommendation. DigiDocService does not provide support for encryption/decryption.

DigiDoc system framework consisting of base libraries, intermediate libraries, webservice and end-user applications such as DigiDoc Portal and DigiDoc Client itself according to the following figure:

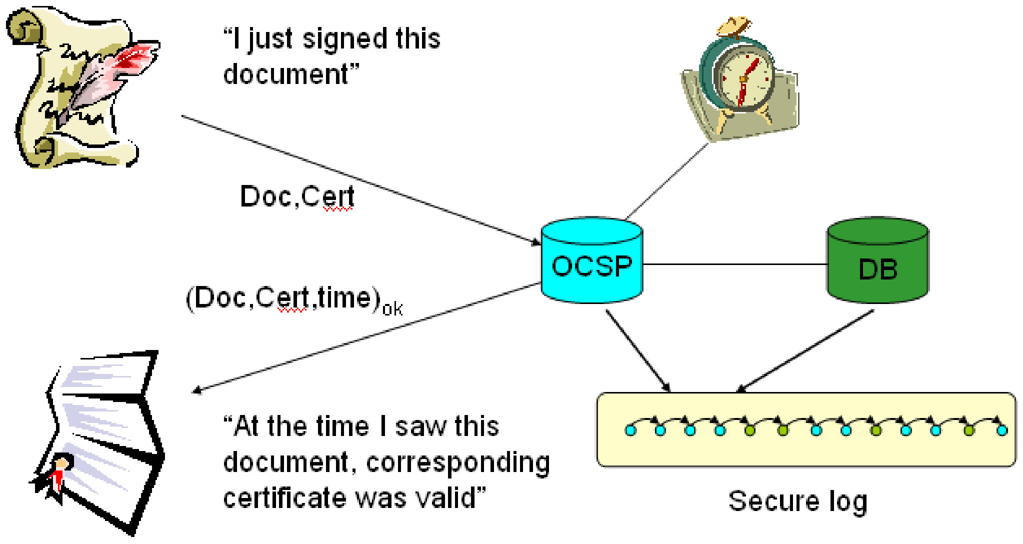


4.2 DigiDoc security model

One of the most challenging issues in digital signature systems is the question of validating a signature long after it was given. Often, the hassle of ensuring validity of signer's certificate at the time of signing is left to the shoulders of verifier.

The DigiDoc and OpenXAdES ideology works the other way – proof of validity of the signer's certificate is obtained at the time of signature creation (or to be precise – immediately after that). This proof is obtained in the format of OCSP response and stored within the signed document.

Furthermore, (hash of the) created signature is sent within the OCSP request and received back within the response. This allows interpreting of the positive OCSP response as "at the time I saw this digitally signed file, corresponding certificate was valid"


"I just signed this document"
Doc,Cert
(Doc,Cert,time)ok
OCSP
DB
"At the time I saw this document, corresponding certificate was valid"
Secure log

The figure above illustrates the essence of the OCSP service acting as a digital e-notary confirming signatures created locally with smartcard.

From infrastructure side this security model requires standard OSCP responder. Hash of the signature is placed on the "nonce" field of the OCSP request structure.

In order to achieve the freshest certificate validity information, it is recommended to run the OCSP responder in "real-time" mode meaning that:

- certificate validity information is obtained from live database rather than from CRL (Certificate Revocation List)
- the time value in the OCSP response is actual (as precise as possible)

To achieve long-time validity of digital signatures, a secure log system is employed within the model. All OCSP responses and changes in certificate validity are securely logged to preserve digital signature validity even after private key compromise of CA or OCSP responder.

It is important to notice that additional time-stamps are not necessary when employing the security model described:

- time of signing and time of obtaining validity information is indicated in the OCSP response
- The secure log provides for long-time validity without need for archival timestamps

4.3 Format of digitally signed file

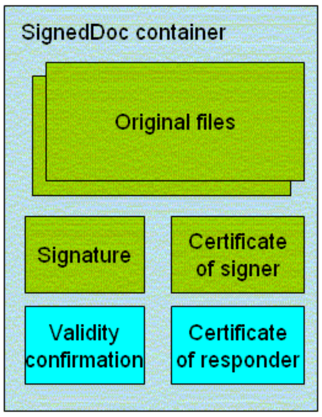
The format of the digitally signed file is based on ETSI TS 101 903 standard called "XML Advanced Electronic Signatures (XAdES)". This standard provides syntax for digital signatures with various levels of additional validity information.

In order to comply with the security model described above, the XAdES profile of "XAdES-X-L" is used in the DigiDoc system but "time-marks" are used instead of "time-stamps" – siging (and certificate validation) time comes with OCSP response.

This profile:

- allows for incorporating following signed properties
    - Certificate used for signing
    - Signing time
    - Signature production place
    - Signer role or resolution
- Incorporates full certificate validity information within the signature
    - OCSP response
    - OCSP responder certificate

As of result, it is possible to verify signature validity without any additional external information – the verifier should trust the issuer of signer's certificate and a OCSP responder certificate.



Original files (which were signed) along with the signature(s), validation confirmation(s) and certificates are encapsulated within container with "SignedDoc" being as a root element.

DigiDoc system uses file extension .ddoc to distinguish digitally signed files according to the described file format.

Syntax of the .ddoc files is described in the separate document in detail.

5 Suggestions and requirements about using the service

5.1 Digital signing

Application provider shall guarantee the following:

- The user is informed about the legal consequences of the digital signature before entering PIN2;
- Measures are implemented to guarantee a single interpretation of signed data;
- The user shall have the possibility to be sure in the authenticity of the signed data and the attributes added to the signature (place of signing, role/resolution) should they be used;
- The data presented to the user before signing is in compliance with the actual data to be signed;
- The user shall have the access to the digitally signed file, which is created after the Digital signing;

5.2 Starting Mobile-ID operations

Mobile-ID operations (mobile authentication and mobile signing) can be started using DigiDocService methods MobileAuthenticate, MobileSign and MobileCreateSignature. All those methods accept Mobile-ID user's Personal Identification Code and/or phone number as input parameter.

Using only phone number:

- Usability – easiest way for users
- Not recommended when security is a concern, because phone numbers are public and Mobile-ID users may get spammed

Using both Personal Identification Code and phone number:

- Spamming is complicated because Personal Identification Codes are not public
- When user makes a mistake when entering either his/her Personal Identification Code or phone number, it's very unlikely that Mobile-ID request will appear in another unintended Mobile-ID user's phone.

It's recommended for application providers to prevent spamming (by IP-restrictions or by using input parameters mentioned above), otherwise Sertifitseerimiskeskus AS must limit access to DigiDocService, to guarentee that DigiDocService stays up and running for other application providers that use it.

Application, that enables users to authenticate or digitally sign documents using Mobile-ID, must clearly present challenge number (ChallengeID parameter in MobileAuthenticate response, see below) to user and warn user to check if challenge number presented by application is the same as challenge number on mobile phone screen. If challenge numbers differ, Mobile-ID operation has to be cancelled.

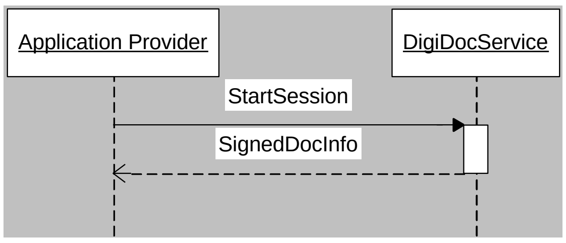
5.3 Technical suggestions and requirements

- Web applications, that enable authentication or digital signing using Mobile-ID or ID-card, should use encrypted channel (HTTPS) between browser and web server.
- Mobile-ID-enabled web applications, when polling regularly DigiDocService with requests about operation state information (whether user has already entered his/her PIN number and signing/authenticating is completed or not), should, for usability reasons, not reload web page every time request is made to DigiDocService – using Ajax is recommended.

6 Main usecases

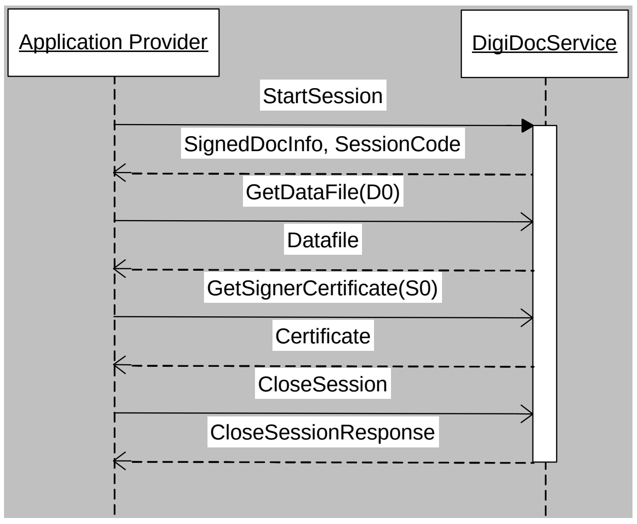
6.1 Verifying a DigiDoc file

In need of verifying a digitally signed document the easiest way is to use the StartSession request (described in chapter 8.1) valuing the SigDocXML parameter. If the only purpose is getting the overview of the content of DigiDoc and no further signing or certificate reading is intended, the StartSession request should be called with the parameter bHoldSession value set to false. In this case no further session closing is neccessary. The StartSession request returns the signed document information as a structure of SignedDocInfo, where all the neccessary parameters the signed document are readable.



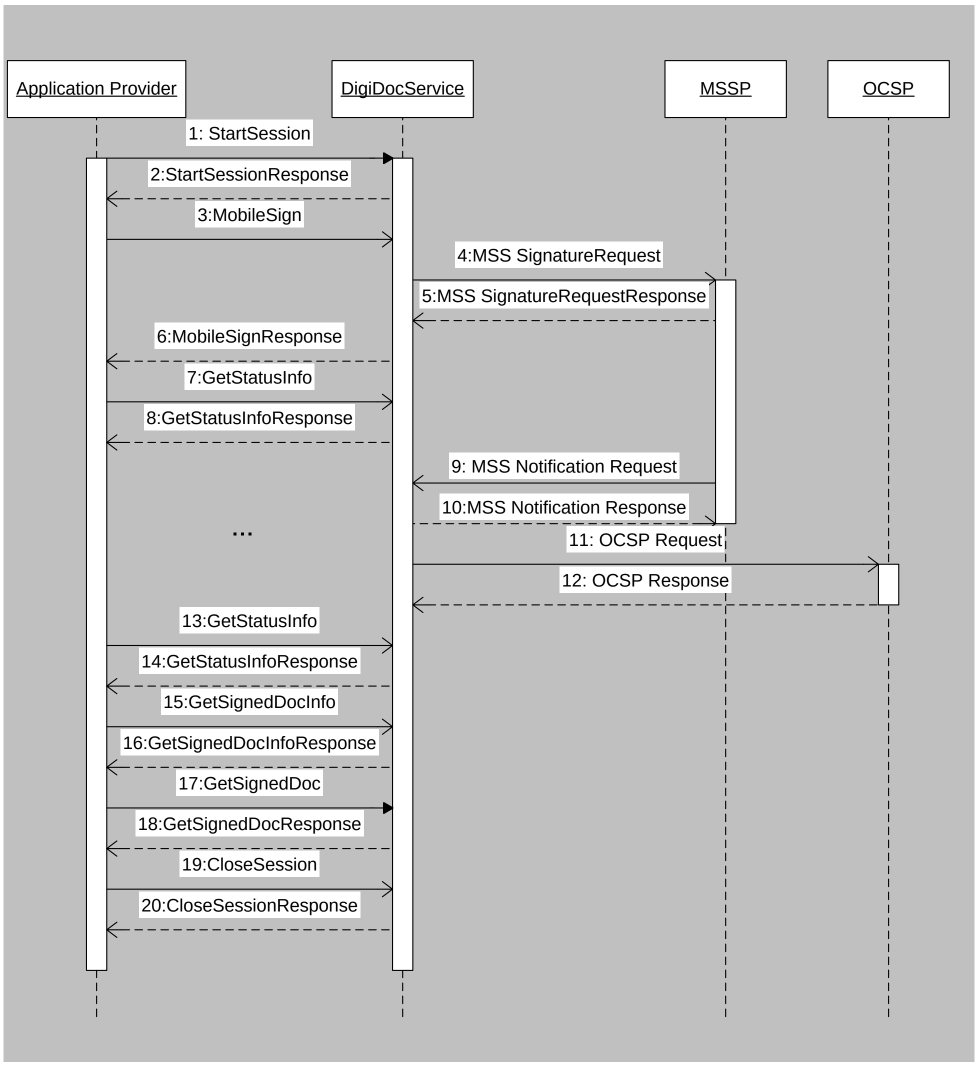
If StartSession is called with parameter bHoldSession=true, after verifying it some additional requests about signed document will be possible:
- to request the information about a data file (GetDataFile method)
- to request the sertificate of a certain signer (GetSignerCertificate method)
- to request the validity confirmation response for a certain signature (GetNotary method)
-to request the validity confirmation signer's certificate of a certain signature (GetNotaryCertificate method)

If StartSession is called with parameter bHoldSession=true, further session closing will be neccessary.



6.2 Signing

6.2.1 *Mobile Signing in Asynchronous Client-Server mode*



1. Application provider sends the files for signing (DigiDoc files or original files) to DigiDoc Service within the StartSession request.
2. As a result of the StartSession request also a created session identifier is returned, what should be used in the headers of following requests.
3. The application sends a MobileSign request to start the signing process. If there's a will to sign more than one original file at a time, it's possible to add additional data files with AddDataFile method before sending the MobileSign request.
4. DigiDocService forwards the signing request to MSSP service, which forwards it in turn to user's phone via a mobile operator.
5. MSSP returns either an errorcode or an information about successful request.
6. DigiDocService returns a response to the application with the MobileSign request. The response is either an errorcode or the information about the signing request.

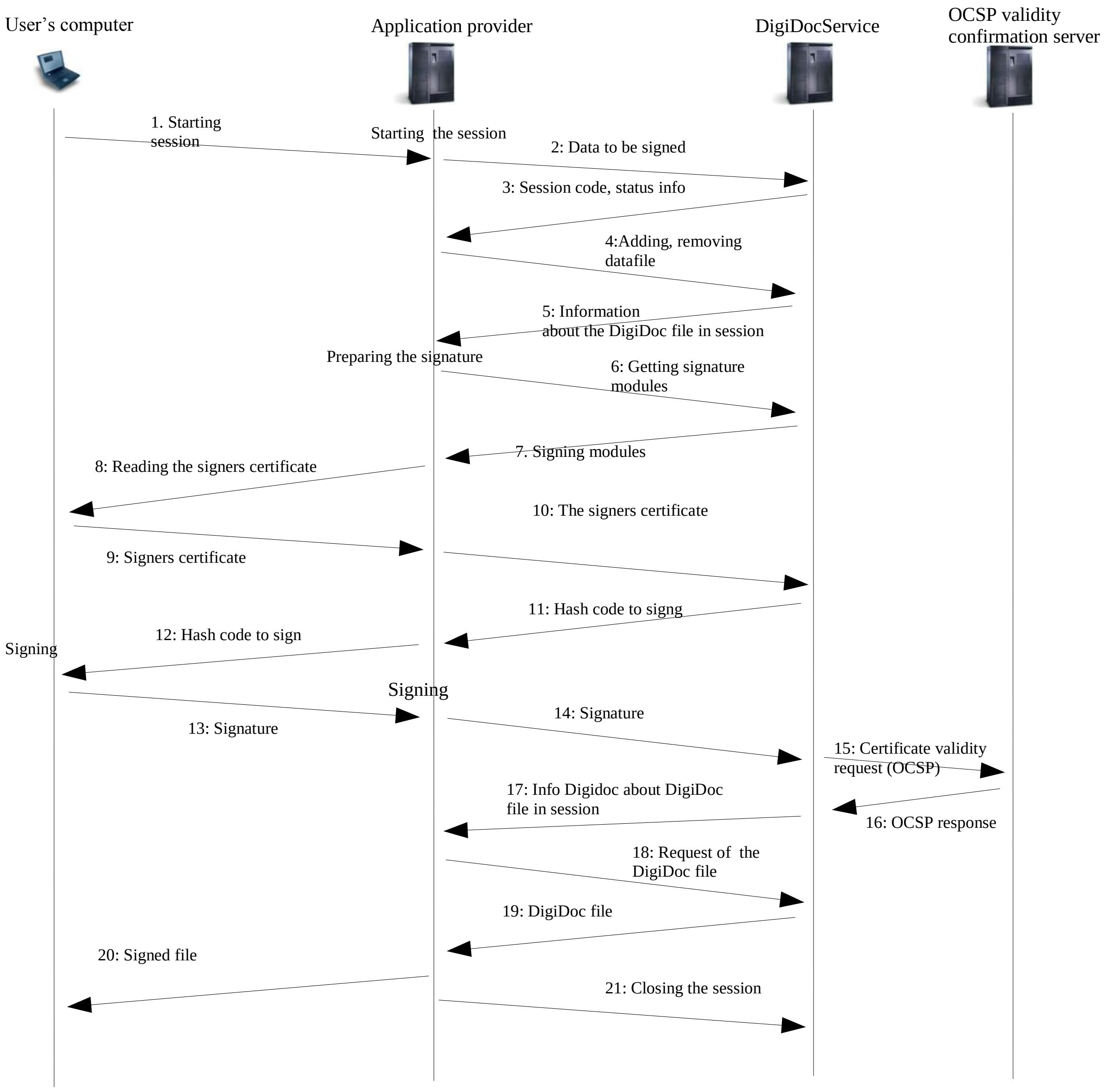
7, 8. In asynchronous Client-Server mode the application should keep up sending a GetStatusInfo request to DigiDocService until signing process is either successful or unsuccessful.

9. MSSP service sends a note about succeeding/unsucceeding. If signing is successful, also a signature will be sent to the DigiDocService.

10. DigiDocService returns the information about receiving the signature to MSSP.
11. After receiving the signature DigiDoc service sends a request about the user certificate's validity to the OCSP validity confirmation service.
12. The validity confirmation service returns a signed validity confirmation response. A signature, which contains a signed hash and the validity confirmation service response is added to the DigiDoc file in session.
13. Another GetSignedDocInfo request is sent by the Application Provider.
14. DigiDocService returns GetStatusInfoResponse about success or failure of signing operation
15. Application provider request information about document status using GetSignedDocInfo method
16. DigiDocService responds to GetSignedDocInfo
17. The application provider inquires the content of the signed DigiDoc with GetSignedDoc request.
18. DigiDocService returns a DigiDoc file to the application. If the content of the data files is not sent to the service within the StartSession, the application that uses the service has to add it to the DigiDoc container itself.
19. The application closes the session with sending a CloseSession request to the service.
20. The Service returns the CloseSession response.

6.2.2 Signing with smartcard

The present example is based on the web-page enabling digital signing.


User's computer
Application provider
DigiDocService
OCSP validity confirmation server
1. Starting session
Starting the session
2: Data to be signed
3: Session code, status info
4:Adding, removing datafile
5: Information about the DigiDoc file in session
Preparing the signature
6: Getting signature modules
7. Signing modules
8: Reading the signers certificate
10: The signers certificate
9: Signers certificate
11: Hash code to signg
12: Hash code to sign
Signing
Signing
13: Signature
14: Signature
15: Certificate validity request (OCSP)
16: OCSP response
17: Info Digidoc about DigiDoc file in session
18: Request of the DigiDoc file
19: DigiDoc file
20: Signed file
21: Closing the session

1. User of the digital signing application has chosen a procedure that requires data signing. The user starts the signing procedure pressing the respective button or hyperlink in a company webservice.

2. The data meant to be signed will be sent to DigiDocService by StartSession request. A new session is initiated with that. Every session is connected to a (digitally signed) document. But every digitally signed document may contain plenty of original files.
   An application sends to the service either
   a. a file to be signed
   b. the metainformation and the hash of the file to be signed (the content of the file has been removed)
   c. the entire conteiner to be signed
   d. the conteiner to be signed without the bodi(es) of datafile(s) (all the content between the DataFile tags has been removed)

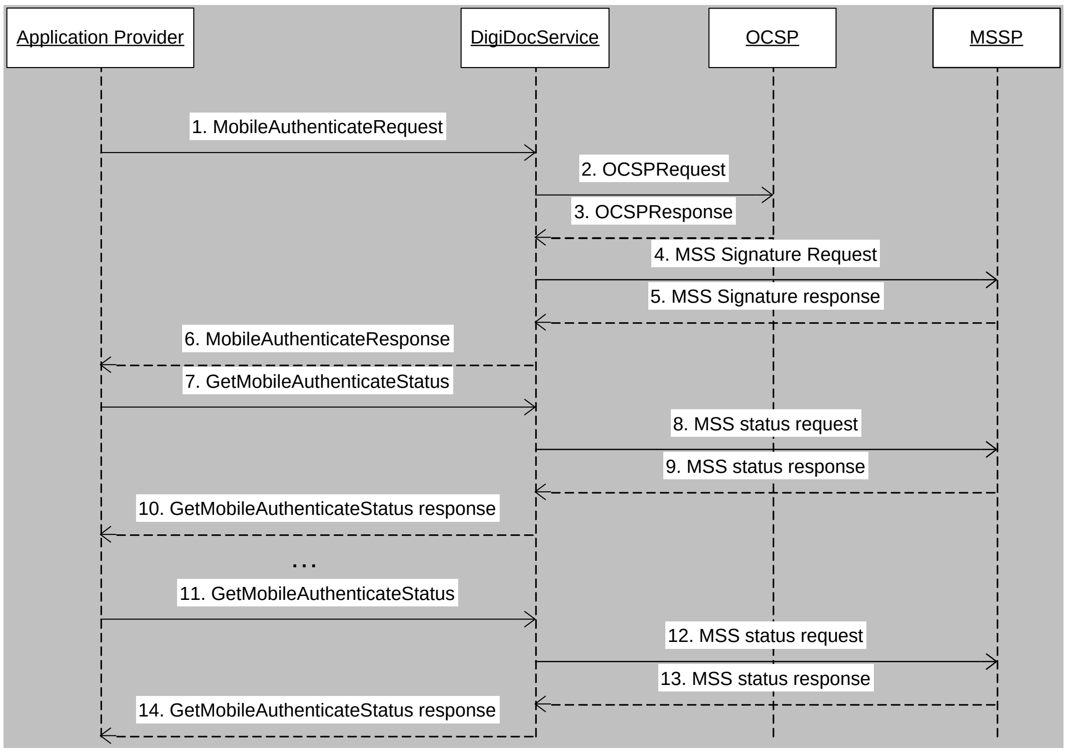
The ways of sending the data necessary for signing are described more precisely in chapter 8.1. Data received within the StartSession request is saved in the session.

3. SessionCode is returned to the application, what enables the following procedures with the session data.
4. Before signing the application may add supplementary datafiles (AddDataFile request) or remove some datafiles (RemoveDataFile request) or carry out some other procedures with session data.
5. After procedures the current session document information is returned.
6. To start the signing process the application provider sends a request to the DigiDocService to get the necessary modules for preparing the signing (GetSignatureModules request). These modules are necessary for webpages to read the signer's certificate from the smartcard (for example from an ID-card) and to forward it to the service.Components in use: ActiveX for IE and Java applet for Mozilla/Netscape/Firefox (the second one also supports the digital signing for Linux/Mac users).
7. The application provider will be returned the necessary signing modules.
8. The signing modules are integrated in the webpage which offers digital signing. Also some information about the signer's role/resolution and the signing location may be asked the user on the webpage. The signing component (ActiveX module or Java applet) located on the webpage reads the signer's certificate information from the smartcard.
9. The certificate from the signer's smartcard together with other user inserted signature attributes is forwarded to the signing web-server.
10. Signature parameters are forwarded to DigiDocService with PrepareSignature request.
11. DigiDocService adds new signature information to the session document – signer's certificate and signature paremeters and calculates the hash, what should be signed by the signer. The signed hash is sent to the application provider in PrepareSignature response.
12. The hash to be signed together with the signing module is displayed to a user. The user presses the signing button on the webpage. As the result of that the signing module signs the hash (also asks for the PIN-code). The created signature is set to the hidden field of the form and sent to the web-page which offers the signing functionality.
13. The signature is forwarded to the signing web-server (application provider).
14. The signature is forwarded to DigiDocServic with FinalizeSignature request.
15. DigiSocService makes a validity confirmation request about the validity of the signer's certificate to the OCSP validity service.
16. OCSP validity confirmation server returns the validity confirmation of the signature.
17. If the confirmation is positive (i.e.the signer's certificate is valid), SK web-service adds the entire information (the signature and the validity confirmation of the signer) to the creatable digital signature. From now on the digital signature is consistent added to the DigiDoc in session. DigiDocService returns the digital signing application the SignedDocInfo.
18. Application asks for the content of the DigiDoc file with GetSignedDoc request.
19. DigiDocService returns the current DigiDoc document which also contains the added signature.

20. The user is informed about the happy end in digital signing. A digitally signed DigiDoc file is ready for download.
    NB! In case that the content of the data file was not sent to servers within StartSession and AddDataFile requests (described in options b and d), it's necessary to add the bodies of data files to DigiDoc file received from the service. The ContentType has to be changed in <DataFile> tag, the reference to hash has to be removed and the contents of data files in Base64 encoding has to be added between <DataFile> tags. If possible the validity of signatures and the integrity of file is checked.
21. The last step for the signing application is to close the session with CloseSession request. After that the service deletes all the data saved within the session.

6.3 Authentication

6.3.1 Mobile authentication in asynchronous Cilent-Server mode


Application Provider
DigiDocService
OCSP
MSSP
1. MobileAuthenticateRequest
2. OCSPRequest
3. OCSPResponse
4. MSS Signature Request
5. MSS Signature response
6. MobileAuthenticateResponse
7. GetMobileAuthenticateStatus
8. MSS status request
9. MSS status response
10. GetMobileAuthenticateStatus response
...
11. GetMobileAuthenticateStatus
12. MSS status request
13. MSS status response
14. GetMobileAuthenticateStatus response

1. The Application Provider sends data required for the authentication to DigiDocService using MobileAutheticate (personal identification code, text to be displayed, language)
2. DigiDocService makes a validity confirmation request about the validity of the user's certificate to the OCSP service.
3. OCSP validity confirmation server returns the validity confirmation of the certificate. If the certificate is valid, go to p. 4, otherwise to p. 6.
4. An authentication request is sent to the user's mobile phone through the MSSP service.
5. MSSP responses with information about successfulness of message delivery to the mobile phone

6. If the certificate was valid and delivery of the authentication message through MSSP was successful, information about the end-user is returned to the Application provider. Otherwise, error message is returned.
7. Application Provider will periodically query the Service with GetMobileAuthenticationStatus request. (Note: this is a case for Asynchronous Client-Server Mode; in other mode the Application Provider will just wait for information from the Service).
8. DigiDocService in turn will query MSSP
9. MSSP responses on status of the query
10. Information about authentication status is forwarded to the Application Provider.
11. This loop (7. 8. 9. 10.) goes on until positive answer or error message will arrive.

6.3.2 Authentication using ID-card

CheckCertificate method can be used as a part of authentication with ID-card, checking the validity of user authentication certificate (located on the ID-card).

7 Queries and Responses for Authentication

All requests and responses are in RPC-encoded style. UTF-8 encoding is used.

7.1 MobileAuthenticate

Query for starting authentication session. First, certificate validity of the user's authentication certificate is verified. In case the certificate is valid, an authentication message is passed to the user's mobile phone. Otherwise, error message is returned. The resulting response to the query contains information about the user, transaction ID and optionally user's certificate for authentication and certificate validity information.

Query:

Parameter	Type	R	Description
IDCode	String	+	Personal Identification Code of the user
Country	String(2)	+	Country of origin. ISO 3166-type 2-character country codes are used (e.g. EE)
PhoneNo	String	-	User's phone number with country code in form +xxxxxxxxx (e.g. +3706234566). In case phone number is given, IDCode and Country parameters are optional. If both PhoneNo and IDCode parameters are given, correspondence between personal code and phone number is verified and in case of inconsistency SOAP error code 301 is returned.
Language	String(3)	+	Language for user dialog in mobile phone. 3-letters capitalized acronyms are used. Possible values: EST, ENG, RUS.
ServiceName	String(20)	+	Name of the service – previously agreed with Application Provider and DigiDocService operator. Maximum length – 20 chars.

MessageToDisplay	String(40)	-	Text displayed in addition to ServiceName and before asking authentication PIN. Maximum length – 40 chars.
SPChallenge	String(20)	-	10-byte random challenge generated by the Application Provider witch would be part of the message for signing by user during authentication. In HEX form. **NB!** For security reasons it's recommended to always fill this parameter with different random value every time.
MessagingMode	String	+	Mode to be used to response for MobileAuthenticate query. Options are: - “asynchClientServer” – Appliaction Provider will make repeated *GetMobileAuthenticateStatus* queries. - “asynchServerServer” – the response will be sent to the Application Provider by in asynchronous mode (see: parameter AsyncConfiguration)
AsyncConfiguration	Integer	-	This parameter is required when using “asynchServerServer” messaging mode and identifies configuration mode. This value has to be previously agreed. Currently, *Java Message Services* (JMS) interface is supported.
ReturnCertData	Boolean	-	If “TRUE”, certificate of the user is returned. Certificate is useful if AP wants to save it and/or independently verify correctness of the signature and validation data.
ReturnRevocationData	Boolean	-	If „TRUE“, OCSP response to the certificate validity query is returned.

Response:

Parameter	**Type**	**Description**
Sesscode	Integer	Session code for current session
Status	String	“OK” if no errors **NB!** “OK” does not mean that the user is successfully authenticated – response “USER_AUTHENTICATED” would indicate this instead. In case error occours, a SOAP error object is returned. Description of the SOAP error object and list of error codes are described in section 7.4.
UserIDCode	String	Personal Identity Code of the user. The value is fetched from “Serial Number” field of the certificate
UserGivenname	String	First name of the user. The value is fetched from “G” (given name) field of the certificate
UserSurname	String	Last name of the user. The value is fetched from “SN” (surname) field of the certificate
UserCountry	String(2)	Country of the origin in ISO 3166 2-character style. The value is fetched from “C” (country) field of the certificate
UserCN	String	„Common Name“ of the certificate holder. The value is fetched from “CN” (common name) field of the certificate

CertificateData	String	User's certificate in BASE64 coding. Returned if parameter ReturnCertData was set „TRUE“ in the query.
ChallengeID	String	4-character control code calculated on basis of the Challenge value to be signed. This code is displayed on mobile phone's screen and shall be also displayed by Application Provider in order to ensure the user on authencity of the query. **NB!** Application provider must ask user to verify that those codes are the same.
Challenge	String	The data to be signed by the user. Consists of mixture of data sent by Application Provider in SPChallenge (10 bytes) field of the query and data added by DigiDocService (also 10 bytes). Returned only if SPChallenge field in the query was set.
RevocationData	String	OCSP response in BASE64 coding. Returned if parameter ReturnRevocationData was set „TRUE“ in the query.

In case asynchClientServer messaging mode is used, the Application Provider shal start sending GetMobileAuthenticateStatus queries until error message or positive answer will be returned.

It is reasonable to wait 15 seconds before starting sending status queries - it is improbable that message from user's phone arrives earlier because of technical and human limitations.

When using asynchServerServer messaging mode, a message is sent to the Application Provider in accordance with previously agreed configuration. The structure of the XML message sent back to Application provider is as follows:

Parameter	**Type**	**Description**
Sesscode	Integer	Session identifier
Status	String	"USER_AUTHENTICATED" in case of successful authentication. Other possible values are described in the description of response to the "GetMobileAuthenticateStatus" query.
Data	String	Signature value in PKCS#1 container in BASE64 encoding. Returned only if SPChallenge field in the query was set.

7.2 GetMobileAuthenticationStatus

This method is relevant when asynchClientServer messaging mode is used.

Query:

Parameter	**Type**	**R**	**Description**
Sesscode	Integer	+	Session identifier – use the value returned with MobileAuthenticate method
WaitSignature	Boolean	+	If „TRUE“ then the Service will wait for response from

			MSSP before responding. „FALSE“ will cause the Service to respond immediately.

Response:

Parameter	Type	Description
Status	String	Process status: - OUTSTANDING_TRANSACTION – authentication is still on the way; - USER_AUTHENTICATED – authentication successful; - NOT_VALID – the action is completed but the signature created is not valid; - EXPIRED_TRANSACTION – timeout; - USER_CANCEL – user cancelled the action; - MID_NOT_READY – the MobileID of the SIM is not yet ready for the operations; - PHONE_ABSENT – phone is switched off or out of coverage; - SENDING_ERROR – other error when sending message (phone is incapable of recieving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - INTERNAL_ERROR – technical error.
Signature	String	Signature value in PKCS#1 container in BASE64 encoding. Returned only if SPChallenge field in the query was set in the MobileAuthenticate request. **NB!** For security reasons it’s recommended for application provider to verify the signature, by using a) value that was signed (Challenge parameter from MobileAuthenticate method response), b) public key from Mobile-ID user’s authentication certificate and c) signature (that is calculated by RSA algorithm).

The session will be terminated unless the Status has value OUTSTANDING_TRANSACTION.

7.3 CheckCertificate

Given method can be used to check the validity of certificates issued by Sertifitseerimiskeskus AS and number of foreign Certification Authorities (URL to list of CA’s: http://www.sk.ee/certs/proxyocsp/)

Query:

Parameter	Type	R	Description
Certificate	String	+	Certificate to be checked for validity, in Base64 format. May include „---BEGIN CERTIFICATE---„ ja

			„---END CERTIFICATE---„ lines (according to PEM format)
ReturnRevocationData	Boolean	-	If TRUE, certificate's validity information is returned on RevocationData field in response.

Response:

Parameter	Type	Description
Sesscode	Integer	Identifier of the created session
Status	String	Certificate's validity information: - GOOD – certificate is valid - REVOKED – certificate has been revoked - UNKNOWN – certificate has never been issued or issuer is unknown - EXPIRED – certificate has been expired - SUSPENDED – certificate has been suspended
UserIDCode	String	Certificate owner's Personal Identification Code. In case certificate has been issued by SK, this value will be taken from certificate subject's serial number field.
UserGivenname	String	Certificate owner's given name, this value will be taken from certificate subject's G (given name) field.
UserSurname	String	Certificate owner's surname, this value will be taken from certificate subject's S (surname) field.
UserCountry	String(2)	Certificate owner's country, this value will be taken from certificate subject's C (country) field. ISO 3166 2-letter country codes are used.
UserOrganisation	String	Certificate owner's organisation, this value will be taken from certificate subject's O (Organisation) field.
UserCN	String	Certificate owner's common name, this value will be taken from certificate subject's CN (Common name) field.
Issuer	String	Certificate issuer's common name, this value will be taken from certificate issuers's CN (Common name) field.
KeyUsage	String	Usage of the (secret) key related to the certificate.
EnhancedKeyUsage	String	Enhanced key usage
RevocationData	String	Certificate's validity information (OCSP service's response) in Base64 format. Returned only if request parameter ReturnRevocationData has been set to TRUE, otherwise empty string is returned.

Response parameters are all UTF-8 encoded.

8 Queries and Responses for Digital Signature

All requests and responses are in RPC-literal style. UTF-8 encoding is used. In most cases a SessionCode parameter should be defined in the SOAP message header.

8.1 StartSession

In most cases the transaction with the webservice is started with using the StartSession method, wherein a data file is sent to the service. Based on the data file a DigiDoc file is intended to create. Also a DigiDoc file itself could be sent to the service for adding the signature, checking the content or separating the data file for reading its content. In the course of the StartSession's query a unique session identifier is returned, what should be added to every procedure called within the transaction.

Startsession request parameters:

- **SigningProfile** – Signature creating and verifying profile. For default setting use "" value. This version of the service ignores the parameter.
- **SigDocXML** – A DigiDoc in XML transformed to HTML-Escaped format. For example "<DataFile>" should be transformed to „&lt;DataFile&gt;".
- **bHoldSession** – A flag that indicates whether the data sent within the StartSession should be stored or the session should be closed deleting all the temporary files straight after response. Possible values "TRUE" and "FALSE".
- **Datafile** - Given parameter enables to send to service a data file within the StartSession request. Based on the file a DigiDoc container is created. For example sending a cv.pdf a cv.ddoc is created, that contains the cv.pdf only. The structure of a datafile element is described in chapter 9.3. in current document. While adding the datafile it's unneccessary to determine the identifier.

NB! It's not allowed to send to the service a data of the SigDocXML and the Datafile at the same time, as they exclude each other.

Parameters of the response:
- **status** – Value „OK" or an error string.
- **sesscode** – Session code used for further requests in the given transaction.
- **SignedDocInfo** –If a StartSession request contains a data file or a DigiDoc file, a SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1 in current document.

In the interests of effectiveness and data privacy it's not recommended to send to service an entire data file, but just the information and the hash of the data file (use the format ContentType = HASHCODE). Hash code can be sent to service in both cases: when sending data to sign and create new DigiDoc container (using request DataFile parameter) and when sending existing DigiDoc container (using SigDocXML parameter). Below there are examples for both cases.

Example – sending hash code instead of full data file to the service for signing

For instance, we intend to digitally sign following 42-bytes long (containing 2 CRLF newlines) text file named test.txt:

```
This is a test file
secondline
thirdline
```

At first, we compose following xml-element, in **canonic*** form, where value „VGhpcyBpcyBhIHRlc3QgZmlsZQ0Kc2Vjb25kbGluZQ0KdGhpcmRsaW5l" is previus datafile Base64 encoded and where there is one newline before </DataFile> ending tag (which is DigiDoc-libraries-specific requirement):

```
<DataFile xmlns="http://www.sk.ee/DigiDoc/v1.3.0#"
ContentType="EMBEDDED_BASE64" Filename="test.txt" Id="D0"
MimeType="text/plain"
Size="42">VGhpcyBpcyBhIHRlc3QgZmlsZQ0Kc2Vjb25kbGluZQ0KdGhpcmRsa
W5l
</DataFile>
```

Assuming, that xml canonisation replaced CRLF (\r\n) newlines with LF (\n) and Base64-encoded datafile is in form of 64-symbols long lines and all values, including attribute values, are UTF8 encoded, we proceed by calculating SHA1 hash over previus <DataFile>..</DataFile> element, including tags. We should get HEX value of „b7c7914ab293811e0f0002932d85860a3b934890", which we convert to binary string (consequential bytes): 0xb7, 0xc7, 0x91, ..., 0x90. And at last we Base64-encode the binary string, which gives us following result: „t8eRSrKTgR4PAAKTLYWGCjuTSJA=".

In PHP progamming language, it would look something like that:

```
base64_encode(pack("H*", "b7c7914ab293811e0f0002932d85860a3b934890"));
```

Now, lets compose a data structure for StartSession method's Datafile parameter and call it $inputData:

```
Filename="test.txt"
MimeType="text/plain"
ContentType="HASHCODE"
Size=42
DigestType="SHA1"
DigestValue="t8eRSrKTgR4PAAKTLYWGCjuTSJA="
```

Now lets send this datastructure to DigiDocService, using StartSession method:

```
StartSession(„", „", TRUE, $inputData);
```

What follows are series of calls to DigiDocService to complete the digital signing process. Lets say we have done everything that's needed and DigiDoc container is signed and ready in the service waiting for us to download it. Now we call service's GetSignedDoc method to get the container.

In the downloaded container, we have to replace xml element <DataFile ... ContentType="HASHCODE" ... Id="D0" ... > ... </DataFile> with the one we previusly composed:

```
<DataFile xmlns="http://www.sk.ee/DigiDoc/v1.3.0#"
ContentType="EMBEDDED_BASE64" Filename="test.txt" Id="D0"
MimeType="text/plain"
```

```
Size="42">VGhpcyBpcyBhIHRlc3QgZmlsZQ0Kc2Vjb25kbGluZQ0KdGhpcmRsa
W5l
</DataFile>
```

For now we should have correct DigiDoc container.

* Information about canonic XML: http://www.w3.org/TR/xml-c14n

Example – sending Digidoc container to the service, replacing full datafile with hash code.

For instance, if we have the following DataFile element in DigiDoc container:

```
<DataFile xmlns="http://www.sk.ee/DigiDoc/v1.3.0#"
ContentType="EMBEDDED_BASE64" Filename="test.txt" Id="D0"
MimeType="text/plain"
Size="42">VGhpcyBpcyBhIHRlc3QgZmlsZQ0Kc2Vjb25kbGluZQ0KdGhpcmRsa
W5l
</DataFile>
```

and we wish to send hash code to the service, not full data file, then we should replace the above xml element with the following:

```
<DataFile xmlns="http://www.sk.ee/DigiDoc/v1.3.0#"
ContentType="HASHCODE" Filename="test.txt" Id="D0" MimeType="text/plain"
Size="42" DigestType="sha1"
DigestValue="t8eRSrKTgR4PAAKTLYWGCjuTSJA="></DataFile>
```

After completing operations (verifying document, or adding signatures etc) with DigiDoc container that we sent to the service, and downloading the container, we have to make the reverse replacement so that DataFile element contains full data file.

8.2 CloseSession

A transaction is closed by the CloseSession request. As the result of the request all the information stored in the server within this session will be deleted. To start a new session a StartSession request should be sent once again. It's always recommended to close a transaction with the CloseSession request. If the application doesn't close the session itself, it will be closed automatically after timeout.

Parameters of the request:

- **sesscode** – An identifier of the active session.

Parameters of the response:

- **status** - If the request is successful, the value will be „OK".

If the request is unsuccessful, a SOAP-FAULT object will be returned.

8.3 CreateSignedDoc

If an application desires to define the format and version of the formable conteiner, the CreateSignedDoc request will be used for creating a new conteiner.
After the CreateSignedDoc request takes place the AddDataFile request for adding the data. Now the file is ready for digital signing.

Parameters of the request:

- **sesscode –** An identifier of the active session.
- **Format –** a format of a creatible document container (currently supported DIGIDOC-XML)
- **Version –** a version number of the format of a creatible document container (currently the supported versions for DIGIDOC-XML are 1.1,1.2,1.3)

The description of DigiDoc formats are available on the webpage http://www.sk.ee/digidoc.

A response of the request includes the following information:

- **status** - If the request is successful, the value will be „OK".
- **SignedDocInfo** – SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

If an inappropriate combination of given format and version number is used in request parameters, a SOAP error object with error message "Invalid format & version combination!" will be returned.

8.4 AddDataFile

AddDataFile request enables to add an additional data file to a DigiDoc container which is in session. If one datafile is added within the StartSession, but the user would like to sign a few more data files in a DigiDoc container, then using this method the rest of the data files will be added before signing.
NB! Adding a data file is possible in the DigiDoc file with no signatures only.

Parameters of the request:

- **Sesscode** – An identifier of the active session.
- **Filename** – Name of the data file without the path.
- **MimeType** – Type of the datafile.
- **ContentType** – Data file's content type (HASHCODE, EMBEDDED_BASE64)
    - **HASHCODE** – To service is sent the hashcode* only, not the entire data file's content. The method how to calculate the hashcode is described in parameter *DigestType* and the hashcode itself is in parameter *DigestValue*
    - **EMBEDDED_BASE64 –** The content of the file is in Base64 encoding in Content parameter.
- **Size** – The actual size of data file in bytes.
- **DigestType** - Hashcode type. So far the SHA1 algorythm is used (parameter value sha1). Required for HASHCODE content type only.

- **DigestValue** – The value of data file's hash* in Base64 encoding.. Required for HASHCODE content type only.
- **Attributes** - Arbitrary amount of other attributes (meta data), what's add to <Datafile> element in DigiDoc file as attributes (in form <name>="<value>").
- **Content** - The content of data file in Base64 encoding, is set if ContentType is EMBEDDED_BASE64.

* See example, how to calculate hash over data file and send it to the service from section 8.1

Response to the request:

- **Status** – If the request is successful, the value will be „OK".
- **SignedDocInfo** – SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

8.5 MobileSign

The MobileSign method invokes mobile signing of a DigiDoc file in the current session. For using the MobileSign method, at least one datafile shall be in DigiDoc container.

In case creation of "pure" mobile signature is needed – i.e. without creating DigiDoc file and/or sending it to the service – MobileCreateSignature should be used instead.

Parameters of the request:

- **Sesscode** – An identifier of the active session.
- **SignerIDCode** – Identification number of the signer (personal national ID number).
- **SignersCountry** – Country which issued the personal national ID number in ISO 3166-style 2-character format (e.g. "EE")
- **SignerPhoneNo –** Phone number of the signer with the country code in format +xxxxxxxxx (for example +3706234566). If both SignerPhoneNo and SignerIDCode parameters are given, correspondence between personal code and phone number is verified and in case of inconsistency SOAP error code 301 is returned.
- **ServiceName** - Name of the service – previously agreed with Application Provider and DigiDocService operator. Required, maximum length – 20 chars.
- **AdditionalDataToBeDisplayed** – Additional text shown to the signer. Optional, maximum length 50 chars.
- **Language** – Language of the message displayed to the signer's phone. ISO 639 a 3-character-code in uppercase is used (for exaple EST, LIT). Required.
- **Role -** The text of the role or resolution defined by the user. Optional.
- **City** – Name of the city, where it's signed. Optional.
- **StateOrProvince** – Name of the state/province, where it's signed. Optional.
- **PostalCode** – Postal code of the signing location. Optional.

- **CountryName** – Name of the country, where it's signed. Optional.
- **SigningProfile –** Specifies the signature creating profile. This parameter gives the service user a possibility to define how the signature should look like. If default SigningProfile is used the value should be "". NB! SigningProfiles are not implemented in the current service version.
- **MessagingMode** - Determines the mode how the response of the MobileSign is returned. Following modes are supported::
  - "asynchClientServer" – Some additional status request are made after MobileSign request by the Application Provider
  - "asynchServerServer" – After signing or in case of an error the server sends a request to the client-application . The client application should be capable to act in server mode to recieve the signature information request according to the parameters in AsyncConfiguration parameter.
- **AsyncConfuguration** – Determines configuration used in asynchServerServer messaging mode. This shall be agreed previously between Application Provider and DigiDocService provider.
- **ReturnDocInfo –** If the value is true, the DigiDoc file information is returned as a result of the request.
- **ReturnDocData –** If the value is true, a DigiDoc document in HTMLescaped format SignedDocData element is returned.

Response to the request:

- **Status –** "OK" or error message.
- **StatusCode** – If the request is successful, 0 is returned, otherwise an error code.
- **ChallengeID –** 4-digit control code calculated from hash of the value to be signed. The control code shall be displayed to the user in order to provide means to verify authencity of the signing request.

If asynchClientServer messaging mode is used then an Application Provider shall start sending GetSignedDocInfo requests to complete the signing session.

It is reasonable to wait ~10 seconds before starting sending status queries - it is improbable that message from user's phone arrives earlier because of technical and human limitations.

In case asynchServerServer messaging mode is used, a message will be sent from DigiDocService acoording to previously agreed configuration. The message is sent in XML format as following:

Parameter	**Type**	**Description**
Sesscode	Integer	An identifier of the active session.
Status	String	Status code. „OK" if no errors, other possible responses are described in description of GetSignedDocInfo request (field „Status").
Data	String	a) XML structure described in section 9.1 of the document if value of the ReturnDocInfo was set true on the request. b) DigiDoc file in if ReturnDocInfo was set "false" and ReturnDocData was set "true" in the request. c) Empty if noth ReturnDocInfo and ReturnDocData

		were set „false“ in the request.

8.6 GetStatusInfo

GetStatusInfo request is for getting the information about the document in session (signed) and it's status. GetStatusInfo request is also used in mobile signing in asynchronous Client-Server mode to get the signing process'es state information.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
ReturnDocInfo	Boolean	+	If the value is „true”, in response SignedDocInfo is set.
WaitSignature	Boolean	+	If the value is „true“, response is not sent before message from mobile phone is recieved or error condition is detected. Otherwise response is sent immediately and signature will be sent with next request.

Response:

Parameter	Type	Description
Status	String	Status of the mobile signing process: - REQUEST_OK – initial message was received; - EXPIRED_TRANSACTION – timeout – the user did not enter the signing PIN during given period of time; - USER_CANCEL – the user refused or cancelled the signing process; - SIGNATURE – signature was created; - NOT_VALID – signature created but not valid; - OUTSTANDING_TRANSACTION – signing in process, please make new request; - MID_NOT_READY – Mobile-ID functionality of the phone is not yet ready; - PHONE_ABSENT – Delivery of the message was not successful, mobile phone is probably switched off or out of coverage; - SENDING_ERROR – other error when sending message (phone is incapable of recieving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - INTERNAL_ERROR – technical error,
StatusCode	String	Status code of the last request
SignedDocInfo	SignedDocInfo	If “ReturnDocInfo” parameter in the GetSignedDocInfo request was set “true” then SignedDocInfo structure will be returned in the format dessribed in chapter 9.1.

8.7 GetSignedDocInfo

The GetSignedDocInfo method shall be used to retrieve status information and the actual (signed) document from the current signing session.

Request:

- **Sesscode** – And identifier of the active session

Response:

- **Status** – “OK” or an error message
- **SignedDocInfo** – XML structure according to the specification in section 9.1 of the document

8.8 GetSignedDoc

A signed document is returned from the webservice within the GetSignedDoc request. The content of the document is in HTMLencoded format. If there’s a will to recieve the document information in structured format in addition to signed document, the GetSignedDocInfo request should be used.

Parameters of the request are:

- **Sesscode** – The one and only parameter for this request is the active session identifier.

Information returned as a response of the request:

- **Status** –„OK” or error string.
- **SignedDocData** – The signed document in the session in XML format. As the XML tags has been transformed to HTML encoded format, therefore a HTMLDecode transduction should be done before saving the file in filesystem or to database.

8.9 GetDataFile

Within GetSignedDoc request a digitally signed document is sent back from webservice. The result is in HTML encoded format. If there is a will to get the structured document information in addition to the document, a GetSignedDocInfo request will be used.

Parameters of the request are:

- **Sesscode** – The one and only parameter for this request is the active session identifier.

Information returned as a response of the request:

- **Status** – If the request is successful, the value will be „OK”.
- **SignedDocData** – The signed document in the session in XML format. As the XML tags has been transformed to HTML encoded format, therefore a HTMLDecode transduction should be done before saving the file in filesystem or to database.

8.10 RemoveDataFile

GetDataFile request is for inquiring an original file out of a digitally signed file.
For instance if a digitally signed file is uploaded to the service within a StartSession request, it will be possible to read out every single original file with GetDataFile request.

Parameters of the request are:

- **Sesscode** – An identifier of the active session.
- **DataFileId** – an identifier of a data file. In Dxx format, where xx stands for the sequence number. DataFileId is readable in SignedDocInfo structure. The structure is returned to the user of the service as a result of the StartSession or GetSignedDocInfo request.

Information returned as a response of the request:

- **Status** – If the request is successful, the value will be „OK".
- **DataFileData** – the original file information in DataFileInfo structure.

The structure of DataFileInfo is described in chapter 9.3. Data files are returned in the same format as they were sent to the service with StartSession or AddDataFile methods. It means that if the service was sent the content of the data file, the current method will return the block of datafile having the content of the data file in Base64 encoding in DfData field. In case that only hash was sent to the service, only the hash is returned by the method.

If you try to inquire a non-existing data file, you'll receive a SOAP error-object with error-message "No such DataFile!".

8.11 RemoveSignature

RemoveSignature request enables to remove a signature from the digitally signed file in session. As a result of the request a SignedDocInfo without the removed signature is returned.

Parameters of the request are:

- **Sesscode** – An identifier of the active session.
- **SignatureId** – the unique identifier of the signature. Identifications of signatures begin with „S" and the sequence number of the signature is followed (for example S0, S1 etc.). Identifications of the signatures of the session document are available in SignedDocInfo structure.
  This structure is returned to the service user for example as a result of the StartSession or GetSignedDocInfo request.

Information returned as a response of the request:

- **status** – If the request is successful, the value will be „OK".
- **SignedDocInfo** – SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

If removing the signature is unsuccessful, a SOAP error-object will be returned with an error-message.
Potential error-messages:

Must supply Signature id! – the identifier of the signatures is unassigned.
No such Signature! – no signature was found for the signature's identifier as a parameter of the request

8.12 GetSignersCertificate

A request for the certificate of the signer. The request allows the service user to read the signer's certificate from a DigiDoc file (to display to the user for example).

Parameters of the request are:

- **Sesscode** – An identifier of the active session.
- **SignatureId** – the unique identifier of the signature. Identifications of signatures begin with „S" and the sequence number of the signature is followed (for example S0, S1 etc.). Identifications of the signatures of the session document are available in SignedDocInfo structure. The structure is returned to the service user for example as a result of the StartSession or GetSignedDocInfo request.

Information returned as a response of the request:

- **status** – If the request is successful, the value will be „OK".
- **CertificateData** – requested certificate as a string in BASE64 encoding (in PEM format)

If returning the certificate is unsuccessful, a SOAP error-object will be returned with an error-message.
Potential error-messages:

Must supply Signature id! – the identifier of the signatures is unassigned.
No such Signature! – no signature was found for the signature's identifier as a parameter of the request

8.13 GetNotarysCertificate

As a result of the request a validity confirmation signer's certificate of the signature is returned (OCSP server's certificate).

Parameters of the request are:

- **Sesscode** – An identifier of the active session.
- **SignatureId** – the unique identifier of the signature. Identifications of signatures begin with „S" and the sequence number of the signature is followed (for example S0, S1 etc.). Identifications of the signatures of the session document are available in SignedDocInfo structure. The structure is returned to the service user for example as a result of the StartSession or GetSignedDocInfo request.

Information returned as a response of the request:

- **status** – If the request is successful, the value will be „OK".
- **CertificateData** – requested certificate as a string in Base64 encoding (in PEM format)

If returning the certificate is unsuccessful, a SOAP error-object will be returned with an error-message.
Potential error-messages:

Must supply Signature id! – the identifier of the signatures is unassigned.
No such Signature! – no signature was found for the signature's identifier as a parameter of the request
No notary for this Signature! – no validity confirmation for the signature as the request of the parameter.

8.14 GetNotary

The request returns the validity confirmation of the certain signature.

Parameters of the request are:

- **Sesscode** – An identifier of the active session.
- **SignatureId** – the unique identifier of the signature. Identifications of signatures begin with „S" and the sequence number of the signature is followed (for example S0, S1 etc.). Identifications of the signatures of the session document are available in SignedDocInfo structure. The structure is returned to the service user for example as a result of the StartSession or GetSignedDocInfo request.

Information returned as a response of the request:

- **Status** – If the request is successful, the value will be „OK".
- **OcspData** – OCSP validity confirmation in Base64 encoding.

If returning the validity confirmation is unsuccessful, a SOAP error-object will be returned with an error-message.
Potential error-messages:

Must supply Signature id! – the identifier of the signatures is unassigned.
No such Signature! – no signature was found for the signature's identifier as a parameter of the request
No notary for this Signature! – no validity confirmation for the signature as the request of the parameter.

8.15 GetVersion

The request enables to check the service and to get to know it's version number. The request has no parameters.

The structure of the answer:

- **Name** – name of the service (currently DigiDocService).
- **Version** – the version of the service in the form of x.x.x (for example 1.0.3) The highest grade stands for major changes in the service, the

second grade describes the changes which may eventuate in changing the protocol of the service. The last grade means some little fixes, which doesn't change the protocol.

- Libname – DigiDoc library name
- Libver – DigiDocLibrary version

8.16 PrepareSignature

The request is used for digital signing preparation if signing with smartcard.
As a result of the request a new so called half-done signature is added to the DigiDoc conteiner in session and the unique identifier of the signature and the hash to be signed is returned. The hash should be forwarded to the the signing software (ActiveX or Java Applet) of the user's computer.

Request:

- **Sesscode** – An identifier of the active session.
- **SignersCertificate** – signer's certificate transferred to HEX string format (from binary (DER) format). Mostly the signing software (ActiveX or Java Applet) in the user's computer delivers the certificate in a proper format.
- **SignersTokenId** – identifier of the private key's slot on a smartcard. The signing software defines it's value within reading the signer's certificate and forwards it to the signing application.
- **Role** **-** The text of the role or resolution defined by the user.
- **City** – Name of the city, where it's signed.
- **State -** Name of the state, where it's signed.
- **PostalCode** – Postal code of the signing location.
- **CountryName** – Name of the country, where it's signed.

Usually the signing application asks the user about the location information of the signing and forwards it to DigiDocService. Inserting the information about role and signing location is voluntary.

Response:

- **status** – If the request is successful, the value will be „OK".
- **SignatureId** – the unique identifier of the signature. Identifications of signatures begin with „S" and the sequence number of the signature is followed (for example S0, S1 etc.). Identifications of the signatures of the session document are available in SignedDocInfo structure. The structure is returned to the service user for example as a result of the StartSession or GetSignedDocInfo request.
- **SignedInfoDigest –** The hash to be signed as a hexadecimal string.

If returning the validity confirmation is unsuccessful, a SOAP error-object will be returned with an error-message.
Potential error-messages:
Must supply Signature certificate! – the value of the signer's certificate is empty.

8.17 FinalizeSignature

The request is used for finalizing the digital signing while signing with smartcard. With FinalizeSignature request the signature prepared at PrepareSignature step is finished. A digitally signed signature is added to DigiDoc file and an OCSP validity confirmation is taken.

Request:

- **Sesscode** – An identifier of the active session.
- **SignatureId** – the unique identifier of the signature. Identifications of signatures begin with „S" and the sequence number of the signature is followed (for example S0, S1 etc.). Identifications of the signatures of the session document are available in SignedDocInfo structure. The structure is returned to the service user for example as a result of the StartSession or GetSignedDocInfo request.
- **SignatureValue –** value of the signature (signed hash) as a HEX string. The signing software (ActiveX or Java Applet) returns the value.

Response:

- **Status –** If the request is successful, the value will be „OK".
- **SignedDocInfo** – SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

8.18 GetSignatureModules

A request for downloading the neccessary modules for digital signing with smartcard. As a result of the request signing modules are sent to the application which uses the service. These modules have to be integrated into the user interface. There are two kinds of modules: FAIL (driver, applet etc) and HTML (dynamical HTML/JavaScript). The code of HTML modules include markers what need to be replaced with respective information (hash, certificate etc.). If neccessary it's possible to do some transductions in HTML code (for example the appearance/the functionality of javascript), but the interchange of parameters between signing applet/ActiveX and the webservice must be guaranteed.

Parameters of the request are:

- **Sesscode** – An identifier of the active session.
- **Platform –** a platform which needs the signing modules
  Options:
  - LINUX-MOZILLA – a Java Applet as a signing application and for Linux the PKCS#11 driver for smartcard communication is returned.
  - WIN32-MOZILLA – a Java Applet as a signing application and for Windows the PKCS#11 module for smartcard communication is returned.
  - WIN32-IE – an ActiveX module as a signing application is returned.
- **Phase** – a step for the modules are being downloaded:
  Options:
  - PREPARE – signing modules for reading the certificate on the smartcard (necessary before calling the PrepareSignature method)
  - FINALIZE – signing modules for signing (necessary before calling the FinalizeSignature method)
- **Type –** defines which modules are returned
  Options:

- FILE – only file-type signing modules are (ActiveX, Applet, PKCS#11 modules) returned
- HTML – only HTML modules are returned
- ALL – both file-type and HTML modules are returned

Information returned as a response of the request:

- **Status** – If the request is successful, the value will be „OK”.
- **Modules** – list of the modules sent as a result of the request. Every element contains the following attributes:
  - Name – name of the module. If it’s a FILE-type module, the module has to be saved in the file with the same name.
  - Type - defines whether it’s a HTML (value of the parameter is “HTML”) or a file-type (value of the parameter is “FILE”) module.
  - Location – Defines the location on the webpage, where the component should be integrated. Options:
    - HTML-HEAD
    - HTML-FORM-BEGIN
    - HTML-FORM-END
    - HTML-BODY
    - HTML-SCRIPT
    - LIBDIR file has to be saved in the directory which has a reference from the HTML page, by default it’s the same directory where you run the script. On the HTML page a module has to be situated in the right place depending on the HTML location.
  - ContentType – defines the format for encoding the content in Content field. Currently the BASE64 encoding is used.

  Data file’s content type (HASHCODE, EMBEDDED_BASE64)
  - Content – content of the module in the form as defined in ContentType parameter.

8.19 GetTSACertificate

The request returns the time stamping authority (TSA) certificate.

This method is useful when signed document contains RFC 3161 timestamps.
In DigiDoc formats 1.0, 1.1, 1.1, 1.2 and 1.3 OCSP responses are treated as timestamps (“timemarks” according to XadES terminology) and RFC 3161 timestamps are not used.

NB! In the current version of service only the interface of the method is implemented, but not the functionality.

8.20 GetTimestamp

The request returns the timestamp in base64 encoding.

This method is useful when signed document contains RFC 3161 timestamps.

In DigiDoc formats 1.0, 1.1, 1.1, 1.2 and 1.3 OCSP responses are treated as timestamps ("timemarks" according to XadES terminology) and RFC 3161 timestamps are not used.

NB! In the current version of service only the interface of the method is implemented, but not the functionality.

8.21 GetCRL

The request returns the certificate revocation list.

This method is useful when signature conatins CRLs for the signature verification. In DigiDoc formats 1.0, 1.1, 1.1, 1.2 and 1.3 only OCSP responses are used.

NB! In the current version of service only the interface of the method is implemented, but not the functionality.

8.22 MobileCreateSignature

This request is used for creating additional signature to the DigiDoc file. The "<Signature>" block is returned as a result and the Application Provider shall take care of inserting this block into DigiDoc file.

The request is built for one-step creation of mobile signature. The method takes care of acquiring of signer's certificate, validity confirmation abd RFC3161-type timestamps if needed in addition to getting mobile signature from the user.

There is no need to create independent session with StartSession method when using MobileCreateSignature method. If session-based procedure is needed, MobileSign method should be used instead.

Request:

Parameter	**Type**	**R**	**Description**
IDCode	String	+	Personal Identification Code of the user
Country	String(2)	+	Country of origin. ISO 3166-type 2-character country codes are used (e.g. EE)
PhoneNo	String	-	User's phone number with country code in form +xxxxxxxxx (e.g. +3706234566). In case phone number is given, IDCode and Country parameters are optional. If both PhoneNo and IDCode parameters are given, correspondence between personal code and phone number is verified and in case of inconsistency SOAP error code 301 is returned.
Language	String(3)	+	Language for user dialog in mobile phone. 3-character capitalized acronyms are used. Possible values: ENG, EST, RUS.
ServiceName	String(20)	+	Name of the service – previously agreed with

			Application Provider and DigiDocService operator. Maximum length – 20 chars.
MessageToDisplay	String(40)	-	Text displayed in addition to ServiceName and before asking authentication PIN. Maximum length – 40 chars.
Role	String	-	Role or resolution of the signature
City	String	-	City where the signature is created
StateOrProvince	String	-	State or province where the signature is created
PostalCode	String	-	Postal code of teh place where the signature is created
CountryName	String	-	Country where the signature is created
SigningProfile	String	-	Parameter for defining signing profile. Currently not implemented, shall be an empty string
Datafiles	List	+	List of the files to be signed. Every element has following fields: • Id – unique identifier for the file, shall start with „D“ following a number • DigestType – hash algorithm identifier. Only „SHA-1“ is supported • DigestValue – hash value of the data file in BASE64 encoding.
Format	String	+	Format identifier for the signed file, shall equal to “DIGIDOC-XML”
Version	String	+	File format version, e.q. “1.3”
SignatureID	String	+	Identifier of the signature. The Application Provider shall detect identifier of the latest signature and increment this value by one. Counting starts from the value „S0“ („S1“, „S2“ etc to follow).
MessagingMode		+	Mode to be used to response for MobileCreateSignature query. Options are: - “asynchClientServer” – Appliaction Provider will make repeated status queries. - “asynchServerServer” – the response will be sent to the Application Provider by DigiDocService. This requires Application Provider to provide interface for recieving these asynchronous responses.
AsyncConfiguratio n	Integer	-	This parameter is required when using “asynchServerServer” messaging mode and identifies configuration mode. This value has to be previously agreed. Currently, *Java Message Services* (JMS) interface is supported.

Response:

Parameter	**Type**	**Description**
Sesscode	Integer	Identificator of the active session
ChallengeID	String	4-character control code calculated on basis of the Challenge value to be signed. This code is displayed on mobile phone’s screen and shall be also displayed by Application Provider in order to ensure the user on authencity of the query.

Status	String	„OK“ when no errors. In case of an error, SOAB error object is returned according to the specification in section 7.4 of the current document.

If asynchClientServer messaging mode is used then GetMobileCreateSignatureStatus query shall be sent after getting a positive response.

It is reasonable to wait ~10 seconds before starting sending status queries - it is improbable that message from user's phone arrives earlier because of technical and human limitations.

In case asynchServerServer messaging mode, a message will be sent to the Application Provider in accordance of previously agreed configuration. This XML message has a afollowing structure:

Parameter	**Type**	**Description**
Sesscode	Integer	Identificator of the current active session
Status	String	Status code which shall be "SIGNATURE" in case of successful signing. Other possible status codes are described in GetMobileSignatureStatus responses.
Data	String	The resulting <Signature> block in pure XML.

8.23 GetMobileCreateSignatureStatus

The method is used to query status information when using asynchClientServer mobile signing mode.

Request:

Parameter	**Type**	**R**	**Description**
Sesscode	Integer	+	Session identifier
WaitSignature	Boolean	+	If „TRUE“ then the Service will wait for response from MSSP before responding. „FALSE“ will cause the Service to respond immediately.

Response:

Parameter	**Type**	**Description**
Status	String	Process status: - REQUEST_OK – the original message was successfully received; - EXPIRED_TRANSACTION – user timeout; - USER_CANCEL – user cancelled the action; - SIGNATURE – signature was successfully created; - OUTSTANDING_TRANSACTION – authentication is still on the way, the status query shall be repeated; - NOT_VALID – the action is completed but the signature created is not valid;

		- MID_NOT_READY – the MobileID of the SIM is not yet ready for the operations; - PHONE_ABSENT – phone is switched off or out of coverage; - SENDING_ERROR – other error when sending message (phone is incapable of recieving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - INTERNAL_ERROR – technical error.
Signature	String	Signature value in PKCS#1 container in BASE64 encoding.

Is the value in Status field is not OUTSTANDING_TRANSACTION then active session is closed after this request.

8.24 GetMobileCertificate

Meetod Mobiil-ID teenuse olemasolu ja sertifikaatide info pärimiseks.

Päring:

Parameeter	Tüüp	K	Kirjeldus
IDCode	String	+	Personal Identification Code of the user
Country	String(2)	+	Country of origin. ISO 3166-type 2-character country codes are used (e.g. EE)
PhoneNo	String	-	User's phone number with country code in form +xxxxxxxxx (e.g. +3706234566). In case phone number is given, IDCode and Country parameters are optional. If both PhoneNo and IDCode parameters are given, correspondence between personal code and phone number is verified and in case of inconsistency SOAP error code 301 is returned.
ReturnCertData	String	-	Determines whether and which certificate(s) to return in the response (status info is returned in any case): "auth" – request for authentication certificate, "sign" – request for certificate for digital signing, "both" – both, "none" – none. "none" is the default value.

Vastus:

Parameeter	Tüüp	K	Kirjeldus
AuthCertStatus	String	+	OK – authentication certificate is active and ready; NOT_ACTIVATED – certificate is not activated; REVOKED – certificate is revoked; SUSPENDED – certificate is suspended.
SignCertStatus	String	+	OK – certificate for digital signing is active and ready; NOT_ACTIVATED – certificate is not activated; REVOKED – certificate is revoked;

			SUSPENDED – certificate is suspended.
AuthCertData	String	-	Authentication certificate in PEM form
SignCertData	String	-	Digital signing certificate in PEM form

If the user does not possess Mobile-ID SIM, SOAP fault is returned in accordance with p 9.4.

9 Data structures

9.1 SignedDocInfo

Presents the structure of a DigiDoc file (container).

- **Format** – File format for the signed container (currently supported DIGIDOC-XML).
- **Version** - The version of a signed file format (1.1, 1.2, 1.3).
- **DataFileInfo –** Information about the files in conteiner. The data structure is described in chapter 4.3 in the current document. A DataFileInfo section may appear 0..n times in an SignedDocInfo section, depending on the number of data files.
- **SignatureInfo** – Contains the info of the signatures in the signed file. This section may appear 0..n times depending on the number of signatures. Contains the following attributes:
    - **Id** – The unique signature's identifier within the current document/transaction. Signatures' indentifications begin with „S" and the signature's sequence number is followed.
    - **Status** – Signature's status information. A signature will be valid, if the value of the attribute is „OK". If a signature is invalid, the value of the attribute will be „Error" and more precise error information is presented in the Error-element.
    - **Error** – Contains the error information discovered during the signature validation check. Contains following attributes:
        - **code** – Error code;
        - **category** – Error category. There are 3 error categories currently:
          TECHNICAL – technical issue;
          USER – issue caused by user;
          LIBRARY – internal error of the DigiDoc library.
        - **description** – Error description in English. Usable error codes and descriptions are the same as in DigiDoc C-library.
    - **SigningTime** – Local time (f.e.time of the signer's computer, time of signing web server) of signing according to the "The W3C note Date and Time Formats" [5]. NB! This is not the official time of signing, the official time is defined in current structure element *Confirmation-> ProducedAt.*
    - **SignerRole** – The role or resolution marked by the signer at signing. Assigned by following attributes:

        - **Certified** – Defines, whether the role has been assigned by the signer itself or by the CA. Only user-defined roles are in use currently, where the parameter value is 0.
        - **Role** – The text of the role or resolution.
    - **SignatureProductionPlace** - The data, belonging to signature's attributes, describes the place of signing. Those fields are not required in signing.
        - **City** – Name of the city, where it's signed.
        - **StateOrProvince** – Name of the state/province, where it's signed.
        - **PostalCode** – Postal code of the signing location.
        - **CountryName** – Name of the country, where it's signed.
    - **Signer** – Information about the signer including the following attributes:
        - **CommonName** – Name of the signer, taken from the signer certificate's Subject field's CN parameter.
        - **IDCode** – Identification number of the signer, taken from the signer certificate's Subject field's Serial Number parameter.
        - **Certificate** – Main information of the certificate used for signing according to the current document's chapter 9.2.
    - **Confirmation** – OCSP validity confirmation's data structure. Every correct and valid signature contains a structure of a validity confirmation. Confirmation section contains the following attributes:
        - **ResponderID** – Distinguish name of the OCSP validity confirmation server (OCSP Responder ID)
        - **ProducedAt** – Validity Confirmation obtaining time according to the "The W3C note Date and Time Formats" [5] (f.e. "2005.09.14T21:00:00Z"). This time is counted as the official signing time.
        - **Responder Certificate** – Certificate of the validity confirmation service (OCSP) server according to the format described in current document chapter 9.2.
        - **Timestamps** – timestamps info (in the current version of service timestamps are not supported)
        - **CRLInfo** – Certificate Revocation Info (in the current version CRL are not supported)

Sample of stucture:

```
<SignedDocInfo xsi:type="d:SignedDocInfo">
   <format xsi:type="xsd:string"></format>
   <version xsi:type="xsd:string"></version>
   <DataFileInfo xsi:type="d:DataFileInfo">
      <Id xsi:type="xsd:string"></Id>
      <Filename xsi:type="xsd:string"></Filename>
      <MimeType xsi:type="xsd:string"></MimeType>
      <ContentType xsi:type="xsd:string"></ContentType>
```

```
        <Size xsi:type="xsd:int">0</Size>
        <DigestType xsi:type="xsd:string"></DigestType>
        <DigestValue xsi:type="xsd:string"></DigestValue>
        <Attributes xsi:type="d:DataFileAtribute">
           <name xsi:type="xsd:string"></name>
           <value xsi:type="xsd:string"></value>
        </Attributes>
     </DataFileInfo>
   <SignatureInfo xsi:type="d:SignatureInfo">
     <Id xsi:type="xsd:string"></Id>
     <Status xsi:type="xsd:string"></Status>
     <Error xsi:type="d:Error">
        <code xsi:type="xsd:int">0</code>
        <category xsi:type="xsd:string"></category>
        <description xsi:type="xsd:string"></description>
     </Error>
     <SigningTime xsi:type="xsd:string"></SigningTime>
     <SignerRole xsi:type="d:SignerRole">
           <certified xsi:type="xsd:int">0</certified>
           <Role xsi:type="xsd:string"></Role>
     </SignerRole>
     <SignatureProductionPlace
si:type="d:SignatureProductionPlace">
        <City xsi:type="xsd:string"></City>
        <StateOrProvince
     xsi:type="xsd:string"></StateOrProvince>
        <PostalCode xsi:type="xsd:string"></PostalCode>
        <CountryName xsi:type="xsd:string"></CountryName>
     </SignatureProductionPlace>
     <Signer xsi:type="d:SignerInfo">
        <CommonName xsi:type="xsd:string"></CommonName>
        <IDCode xsi:type="xsd:string"></IDCode>
        <Certificate xsi:type="d:CertificateInfo">
           <Issuer xsi:type="xsd:string"></Issuer>
           <Subject xsi:type="xsd:string"></Subject>
           <ValidFrom xsi:type="xsd:string"></ValidFrom>
           <ValidTo xsi:type="xsd:string"></ValidTo>
           <IssuerSerial
           xsi:type="xsd:string"></IssuerSerial>
           <Policies xsi:type="d:CertificatePolicy">
              <OID xsi:type="xsd:string"></OID>
              <URL xsi:type="xsd:string"></URL>
              <Description
                 xsi:type="xsd:string"></Description>
           </Policies>
        </Certificate>
     </Signer>
     <Confirmation xsi:type="d:ConfirmationInfo">
        <ResponderID xsi:type="xsd:string"></ResponderID>
        <ProducedAt xsi:type="xsd:string"></ProducedAt>
        <ResponderCertificate xsi:type="d:CertificateInfo">
           <Issuer xsi:type="xsd:string"></Issuer>
           <Subject xsi:type="xsd:string"></Subject>
           <ValidFrom xsi:type="xsd:string"></ValidFrom>
```

```
            <ValidTo xsi:type="xsd:string"></ValidTo>
            <IssuerSerial
            xsi:type="xsd:string"></IssuerSerial>
            <Policies xsi:type="d:CertificatePolicy">
               <OID xsi:type="xsd:string"></OID>
               <URL xsi:type="xsd:string"></URL>
               <Description
                  xsi:type="xsd:string"></Description>
            </Policies>
         </ResponderCertificate>
      </Confirmation>
     </SignatureInfo>
   </SignedDocInfo>
```

9.2 CertificateInfo

Data structure which includes the main fields of the certificate. Used for describing the information of the certificate of the signer and the information of the certificate of the validity confirmation.

Contains the following attributes:

- **Issuer** – The distinguished name of the certificate issuer.
- **IssuerSerial** – The certificate's serial number.
- **Subject** – The distinguished name of the certificate.
- **ValidForm** – The certificate's period of validity according to The W3C note Date and Time Formats *[5]* (*for example* "2005.09.14T21:00:00Z").
- **ValidTo** – The expiration time of the certificate according to The W3C note Date and Time Formats *[5]*
- **Policies** – Structure of signing policies, may appear 0..n times.
    - **OID** – The unique identifier of signing policies.
    - **URL** – The reference to signing policies (used on company certificates primely).
    - **Description –** A short description of signing policies.

Sample of stucture:

```
<Certificate xsi:type="d:CertificateInfo">
   <Issuer
xsi:type="xsd:string">/emailAddress=pki@sk.ee/C=EE/O=AS
Sertifitseerimiskeskus/OU=ESTEID/SN=1/CN=ESTEID-SK</Issuer>
   <Subject xsi:type="xsd:string">/C=EE/O=ESTEID/OU=digital
signature/CN=KESKEL,URMO,38002240232/SN=KESKEL/GN=URMO/serialNumbe
r=38002240232</Subject>
   <ValidFrom
xsi:type="xsd:string">2005.03.18T22:00:00Z</ValidFrom>
   <ValidTo
xsi:type="xsd:string">2008.03.22T22:00:00Z</ValidTo>
   <IssuerSerial
xsi:type="xsd:string">1111128454</IssuerSerial>
   <Policies xsi:type="d:CertificatePolicy">
```

```
      <OID
      xsi:type="xsd:string">1.3.6.1.4.1.10015.1.1.1.1</OID>
      <URL xsi:type="xsd:string">http://www.sk.ee/cps/</URL>
      <Description xsi:type="xsd:string">none</Description>
   </Policies>
</Certificate>
```

9.3 DataFileInfo

The given data structure describes the information of the data file(s) inside DigiDoc. The structure may contain a data file in Base64 format or just a hash of the data file depending on the value of the ContentType attribute.

- **Id** – Internal unique identifier of a file.The identifiers consist of „D” and the file's sequence number after that. Within a StartSession request the given attribute is not valued and an empty string is sent/forwarded.
- **Filename** – A name of the data file without a path.
- **ContentType** – Data file's content type (HASHCODE, EMBEDDED_BASE64)
    - **HASHCODE** – To service is sent the hashcode* only not the entire data file's content. The method how to calculate the hashcode is described in parameter *DigestType* and the hashcode itself is in parameter *DigestValue*.
    - **EMBEDDED_BASE64 –** The content of the file is in Base64 encoding in DfData attribute.
- **MimeType** – Mime type of file.
- **Size** – The actual size of file in bytes.
- **DigestType** - Hashcode type of the data file. So far only the "sha1" algorithm is supported. Required for HASHCODE content type only.
- **DigestValue** – The value of data file's hash* in Base64 encoding. Required for HASHCODE content type only.
- **Attributes** - Arbitrary amount of other attributes (meta data), what's add to <Datafile> element in DigiDoc file as attributes (in format <name>="<value>").

DfData - Data file content in Base64 encoding.

* See example, how to calculate hash over data file and send it to the service from section 8.1

9.4 SOAP Error Messages

The SOAP error message contains error code in the <faultstring> object and additional text in the <detail><message> object.

Error messages are grouped as follows:

100-199 – errors caused by user (Application Provider) of the service
200-299 – internal errors of the service
300-399 – errors caused by end user and his/her mobile phone

List of error codes:

Error Code	Explanation
100	General error
101	Incorrect input parameters
102	Some of required input parameters is missing
103	Access denied (for example *ServiceName* is not registered with DigiDocService provider)
200	General error of the service
201	Missing user certificate
202	Unable to verify certificate validity
300	General error related to user's mobile phone
301	Not a Mobile-ID user
302	The certificate of the user is not valid (OCSP said: REVOKED)
303	Certificate is not activated or/and status of the certificate is unknown (OCSP said: UNKNOWN)

10 Service Change History

10.1 Differences between versions 2.3.30 and 2.3.5

- Added method CheckCertificate
- Many service inner updates

10.2 Differences between versions 2.3.3 and 2.3.5

- Added method GetMobileCertificate.

10.3 Differences between versions 1.100 and 2.3.3

- Added methods MobileAuthenticate, GetMobileAuthenticateStatus, MobileCreateSignature ja GetMobileCreateSignatureStatus;
- Updated method MobileSign;
- All parameter names are capitalized (Sesscode, Datafile etc).
- Enhanced service performance and continuity

10.4 Differences between versions 1.100 and 1.101

- Added new parameter SigningProfile to methods StartSession, MobileSign, PrepareSignature.
- Session identifier moved from SOAP message header to message body as a first parameter.
- Added methods to deal with RFC 3161 timestamps and certificate revocation lists (CRL).

10.5 Differences between versions 1.000 and 1.100

- Added methods for signing with Mobile-ID: MobileSign ja GetStatusInfo.