AS Sertifitseerimiskeskus

DigiDocService specification

Document version:	**3.8.1**
Last update:	**09.03.2015**
Service version:	**3.8.1**

Table of contents

1 Document Versions....3
2 References....5
3 Terms and Acronyms....6
4 Introduction....6
4.1 Formats of digitally signed files....7
4.1.1 DDOC format....7
4.1.2 BDOC format....8
4.2 Signing algorithms....8
5 Suggestions and requirements for Application providers....9
5.1 Digital signing....9
5.2 Starting Mobile-ID operations....9
5.3 Technical suggestions and requirements....10
6 Main use cases....11
6.1 Verification of the digitally signed file....11
6.2 Signing....13
6.2.1 Mobile Signing in Asynchronous Client-Server mode....13
6.2.2 Signing with smartcard....15
6.3 Authentication....17
6.3.1 Mobile authentication in asynchronous Client-Server mode 17
6.3.2 Authentication using smartcard....18
7 Queries and Responses for Authentication....18
7.1 MobileAuthenticate....18
7.2 GetMobileAuthenticateStatus....21
7.3 CheckCertificate....22
8 Queries and Responses for Digital Signature....23
8.1 StartSession....23
8.1.1 HASHCODE....25
8.2 CloseSession....29
8.3 CreateSignedDoc....30
8.4 AddDataFile....31
8.5 MobileSign....32
8.6 GetStatusInfo....34
8.7 GetSignedDocInfo....35
8.8 GetSignedDoc....36
8.9 GetDataFile....36
8.10 RemoveDataFile....37
8.11 RemoveSignature....38
8.12 GetSignersCertificate....38
8.13 GetNotarysCertificate....39
8.14 GetNotary....40
8.15 GetVersion....40
8.16 PrepareSignature....41
8.17 FinalizeSignature....42
8.18 MobileCreateSignature....43
8.19 GetMobileCreateSignatureStatus....46

8.20 GetMobileCertificate 47
8.21 MobileSignHash 48
8.22 GetMobileSignHashStatusRequest 49
9 Data structures 50
9.1 SignedDocInfo 50
9.2 CertificateInfo 54
9.3 DataFileInfo 54
9.4 SOAP Error Messages 55
9.5 Container validation 57
10 Service Change History 57

1 Document Versions

Ver.	Date	Author	Notes
3.8.1	09.03.15	Risto Alas	- Method “MobileCreateSignature” now supports the BDOC-TS (ASiC-E) format. See the parameter “SigningProfile”, value “LT”. - Added timeout durations for Mobile-ID operations
3.7.1	08.12.14	Risto Alas	- Updated the signing algorithms chapter 4.2: new Mobile-ID SIM cards will support more than one signing algorithm (e.g. both RSA and ECDSA). - Elaborated on ECDSA signature verification for GetMobileAuthenticateStatus() - Introduced a new reserved value “LT” (*Long Term,* based on ASiC-E standard) for "SigningProfile" parameter. The value will be used in a future release for the BDOC-TS (BDOC with time-stamps / ASiC-E) format, currently service outputs an error.
3.6	25.08.14	Risto Alas, Priit Reiser	- Added a description of the BDOC HASHCODE format - The MobileCreateSignature operation has a new optional attribute “mime-type” - Data file content type has been fixed in chapter 9.3 (DataFileInfo)
2.128	17.04.14	Tauri Neitov	- Updates to the descriptions of StartSession() and MobileCreateSignature() - Updated container validation info
2.127	26.03.14	Tauri Neitov	- Updates to the BDOC version 2.1 support - Added a chapter about error messages from the DigiDoc library - Updated description for the

			GetMobileCertificate() response - Elaborated on the support of ECDSA certificates in the response of the operation GetMobileAuthenticateStatus()
2.126	16.01.14	Ago Vesmes, Tauri Neitov	- Corrections to the text. - Updated MobileAuthenticate(), MobileSign(), MobileCreateSignature() and GetMobileCertificate() method descriptions in relation to the mandatory country field. - Added descriptions of the methods MobileSignHashRequest() and GetMobileSignHashStatusRequest(). - Updated descriptions of the SOAP error messages. - Corrected variable name in the method MobileAuthenticate() input. - Updated list of the permitted container formats and versions. - List of the changes to the service is relocated to the website www.id.ee. - Chapters about DigiDoc, Security model of DigiDoc and GetSignatureModules have been removed.
2.125	18.03.13	Ahto Jaago, Liisa Lukin, Ago Vesmes	- Updated MobileAuthenticate(), MobileSign(), MobileCreateSignature() and GetMobileCertificate() method descriptions - Personal Identity Code and phone number are mandatory. Added language parameter value „LIT“. Updated info about the length of MessageToDisplay parameter - Updated the requirements for using Mobile-ID operations - Updated StartSession() method description - Adding signature and creating new container only DIGIDOC-XML 1.3 format is supported - Updated MobileCreateSignature() and CreateSignedDoc() methods descriptions. - Updated GetMobileCreateSignatureStatus() method description - Updates 6.2.2 Signing with smartcard - added a recommendation to use the idCard.js client side library instead of the GetSignatureModules operation. - Added info about file size limit and HASHCODE in section 5.3, 8.1, 8.4, 8.18 - Updated error codes in section 9.4, 8.6, 8.20 - Added description of differences between Service versions 2.3.5 and 3.2.5.
2.123	19.12.08	Ahto Jaago,	- Added section „Suggestions and

		Urmo Keskel	requirements about using the service“ - Added CheckCertificate method description - Added section „Authentication using ID-card“ - Updated descriptions of following methods and data structure: StartSession, MobileAuthenticate, MobileAuthenticateStatus, AddDataFile, DataFileInfo - Updated section 6.2.1
2.122	23.04.07	Urmo Keskel	GetMobileCertificate method added, general text corrections
2.120	20.03.07	Urmo Keskel	Major upgrade
1.105	03.05.06	Urmo Keskel	Changed methods StartSession, MobileSign and PrepareSignature, added parameter signatureProfile. Added methods describing timestamps and certificate revocations lists.
1.104	07.02.06	Urmo Keskel	Added SmartCard signing functions, SessionCode parameter moved from header to body. Removed exampled.
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.

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.id.ee/28737
[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
[8] NIST P-256	National Institute of Standards and Technology defined P-256 curve http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
[9] BDOC format specification	http://sk.ee/repository/bdoc-spec21.pdf
[10] XMLDSig	XML Signature Syntax and Processing Version 1.1, W3C Recommendation 11 April 2013. http://www.w3.org/TR/xmldsig-core1/

3 Terms and Acronyms

Application Provider	Client of the DigiDocService, provides an application that uses digital signing, signature verification and/or authentication.
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 authenticity 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 mobile network to the phone and the user shall enter PIN code to perform transaction. The signed result is sent back to the service.
MSSP	Mobile Signature Service Provider. Described in standard 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 request.
Verification	Checking the validity of signatures of the digitally signed data.
Transaction, session	Communication while a file (DigiDoc or the original data file) is forwarded to the web service and some operations related to these are followed, i.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 web service enabling an 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 (including any smartcard)
- Creation of DigiDoc/BDOC files
- Digital signing of DigiDoc/BDOC with Mobile-ID
- Digital signing of DigiDoc/BDOC with ID card (and other smartcards)

- Verification of digitally signed files (DigiDoc/BDOC) and validity of signatures
- Hash signing with Mobile-ID.

Access to the service is created on the basis of an IP address. A contract with Sertifitseerimiskeskus needs to be signed for using the service. The price of using the DigiDocService depends on the number of signature and authentication queries per month and on the number of concurrent queries coming from one application.

DigiDocService supports DigiDoc container formats DIGIDOC-XML 1.3 and BDOC 2.1 *with time-marks*. BDOC *with time-stamps* (BDOC-TS, ASiC-E) format is supported in version 3.8 only for using MobileCreateSignature method. Other methods will support BDOC-TS file format in service version 3.9.

Support for BDOC *with time-stamps* will be added 2015.

Older formats (SK-XML 1.0, DIGIDOC-XML 1.1 and DIGIDOC-XML 1.2) are only supported for verification (document container format is checked in the MobileCreateSignature and CreateSignedDoc methods). If an inappropriate combination of given format and version number is used in request parameters, a SOAP error object with error message “Invalid format and version combination” will be returned.

4.1 Formats of digitally signed files

4.1.1 DDOC format

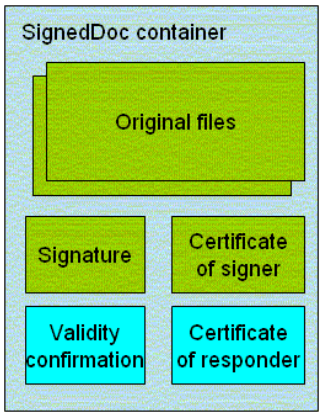
The format of the digitally signed file is based on ETSI TS 101 903 standard called “XML Advanced Electronic Signatures (XAdES)”. This standard provides syntax for digital signatures with various levels of additional validity information.

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” – signing (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 [3] DigiDoc Format Specification in detail.

4.1.2 BDOC format

In addition starting from version 3.5 DigiDocService also supports BDOC 2.1 *with time-marks* (BDOC-TM). BDOC *with time-stamps* (BDOC-TS, ASiC-E) format is supported in version 3.8 only for using MobileCreateSignature method. Other methods will support BDOC-TS file format in service version 3.9.

The description of BDOC file format is available in BDOC specification [9].

Starting from 2015 BDOC is default digital signature format in Estonia, therefore it’s important to add BDOC file format support to your service. For more information about BDOC file format, please visit http://www.id.ee/?id=34336

Instructions for DigiDocService servicee users on how to migrate to BDOC format are available at id.ee website: http://www.id.ee/?lang=en&id=37072 .

4.2 Signing algorithms

DigiDocService supports signatures using the ECDSA (Elliptic Curve Digital Signature Algorithm) and RSA algorithms.

The service automatically chooses the appropriate algorithms for signing and authentication. To learn which algorithm is used in a particular case, application providers should inspect the certificate returned by the service.

ECDSA is currently only supported for Mobile-ID. If user’s SIM card does not have ECDSA support, RSA algorithm is used. For DDOC file format only RSA is supported, BDOC format supports RSA and ECDSA.

A single signer can have multiple active certificates, each with a different signing algorithm. In such cases, DigiDocService chooses the most suitable certificate automatically.

The choice is based on following conditions:

- if user’s SIM-card supports ECDSA, authentication (operation MobileAuthenticate) is always done using the ECDSA certificate.

- If user's SIM-card supports ECDSA and RSA, then ECDSA is used for signing BDOC files (operation MobileSign and MobileCreateSignature). As the DDOC file format does not support ECDSA, DDOC files are always signed using RSA.

For SIM cards that support both ECDSA and RSA the GetMobileCertificate method returns the ECDSA certificate; similarly, the MobileSignHash method chooses ECDSA.

RSA is commonly used with either 1024-bit or 2048-bit keys. ECDSA is implemented over the NIST P-256 [8] curve and the signatures are encoded according to the XMLDSig specification [10] (i.e., two 256-bit integers appended to each other, zero-padded on the left if necessary; the result is then converted to Base64). The total raw ECDSA signature size is always 512 bits.

5 Suggestions and requirements for Application providers

5.1 Digital signing

Application provider shall guarantee the following:

- According to the Digital Signature Act passed in the Estonian parliament, a digital signature solution must make it possible to:
  1) unambiguously identify the person who owns the signing certificate;
  2) identify the time of signing;
  3) connect the digital signature with the signed data in a way that makes it impossible to undetectably change the signed data or its meaning after signing.
- The user is informed about the legal consequences of the digital signature before entering PIN2 (i.e., the PIN used for digital signatures).
- 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 access to the digitally signed file which is created after the digital signing. Note that this applies to any operation that uses PIN2. For example, when signing online payments on a web site, the signer must be allowed to access the signed container after signing. This allows the signer to verify contents of the signed data.

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 phone number as input parameter.

If you would like to provide Mobile-ID operations for Lithuanian Mobile-ID users in your application then both input parameters are mandatory: user's Personal Identification Code and phone number. Otherwise the request fails.

NB! It is highly recommended to use both input parameters - Mobile-ID user's Personal Identification Code and phone number also for Estonian Mobile-ID users. The requirement is planned to be turned obligatory in the future.

Using only phone number is 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:

- 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.
- Spamming is complicated because Personal Identification Codes are not public
- The user does not necessarily have to enter such data directly: for example, a user name could be tied to a particular combination of personal identification code and a phone number.

It's mandatory for application providers to prevent spamming (by IP-restrictions or by using input parameters mentioned above), otherwise AS Sertifitseerimiskeskus must limit access to DigiDocService, to guarantee 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.

Challenge number should be correctly implemented and highly visible also when Mobile-ID operations are used from Mobile device browser.

5.3 Technical suggestions and requirements

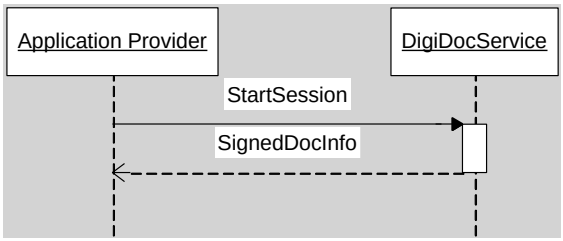
- Web applications, that enable authentication or digital signing using Mobile-ID or ID-card, should use encrypted channel (HTTPS) between browser and web server.
- Mobile-ID-enabled web applications, when polling regularly DigiDocService with requests about operation state information (whether user 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.

- The file size is limited to 4 MB. Section 8.1 describes how to send larger-scale files to the Service.

6 Main use cases

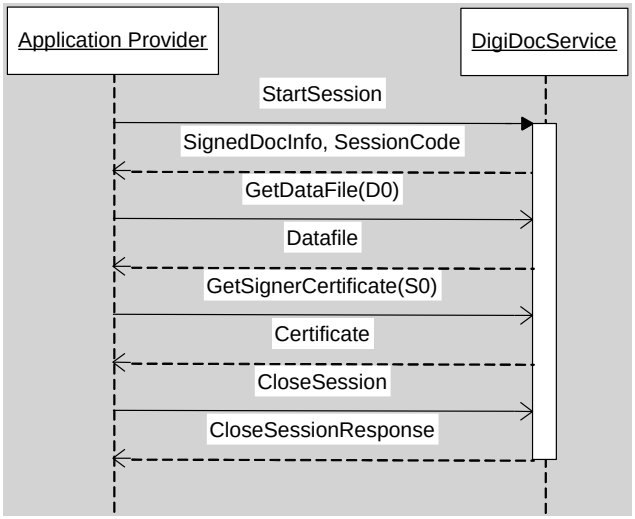
6.1 Verification of the digitally signed file

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 necessary. The StartSession request returns the signed document information as a structure of SignedDocInfo, where all the necessary parameters the signed document are readable.



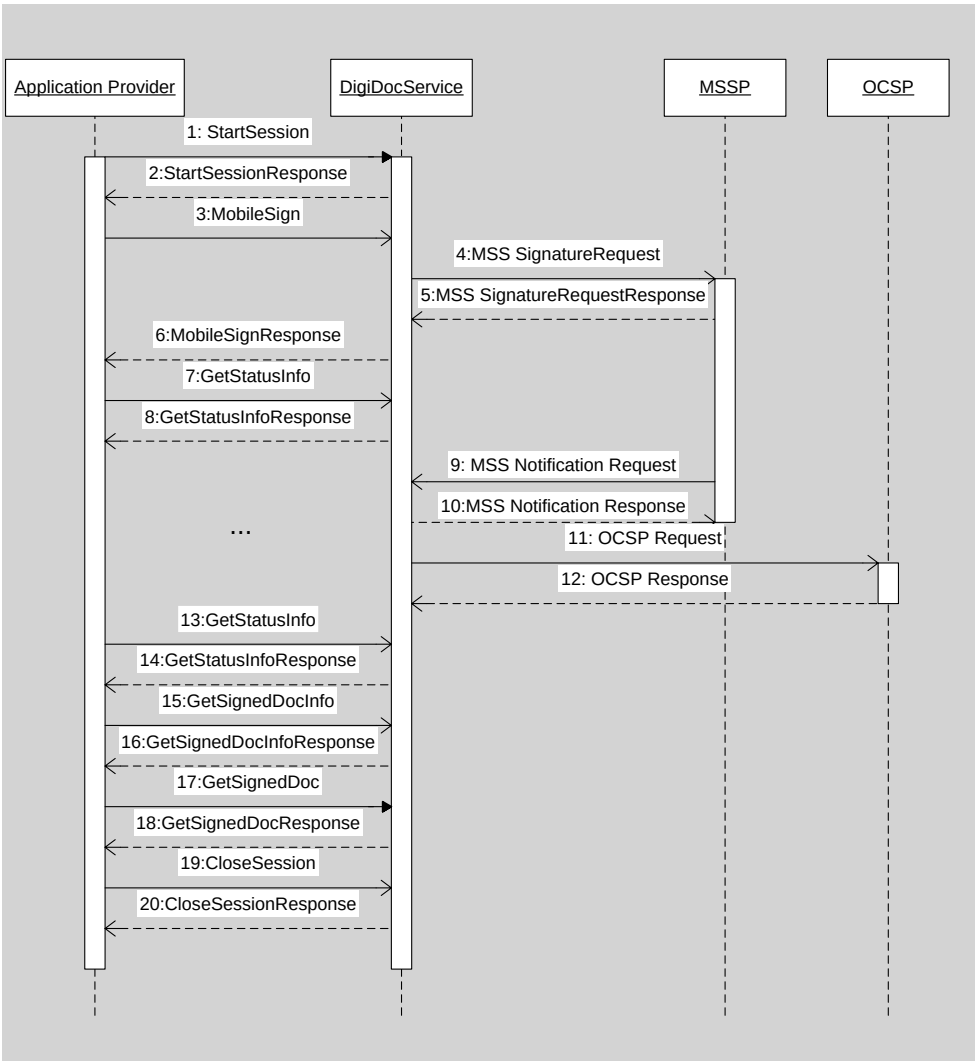
If StartSession is called with parameter bHoldSession=true, after verifying it some additional requests about signed document will be possible:
- to request the information about a data file (GetDataFile method)
- to request the certificate of a certain signer (GetSignerCertificate method)
- to request 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 necessary.


Application Provider
DigiDocService
StartSession
SignedDocInfo, SessionCode
GetDataFile(D0)
Datafile
GetSignerCertificate(S0)
Certificate
CloseSession
CloseSessionResponse

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.

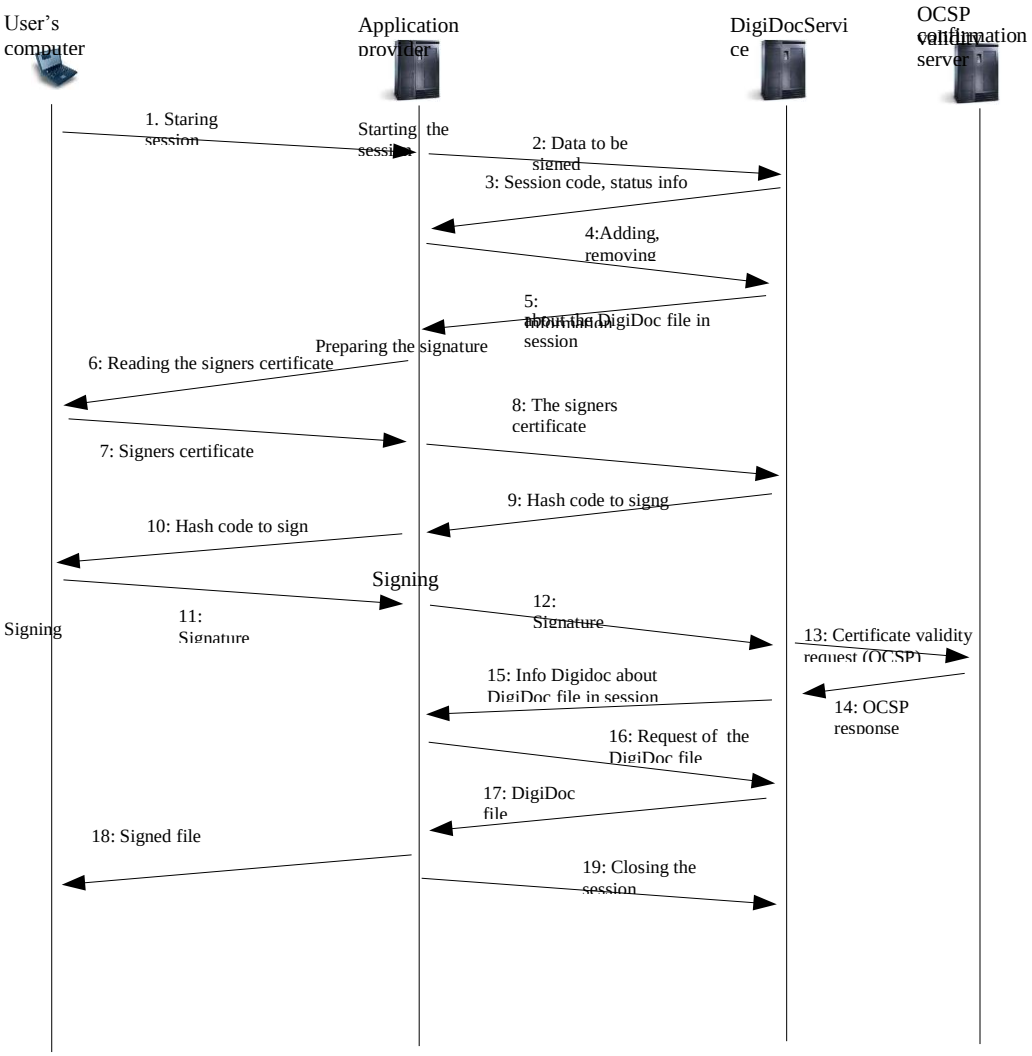
18. DigiDocService returns a DigiDoc file to the application. If the content of the data files is not sent to the service within the StartSession, the application that uses the service has to add it to the DigiDoc container itself.

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.



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 web service.

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 meta information and the hash of the file to be signed (the content of the file has been removed)
   c. the entire container to be signed

d. the container 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.

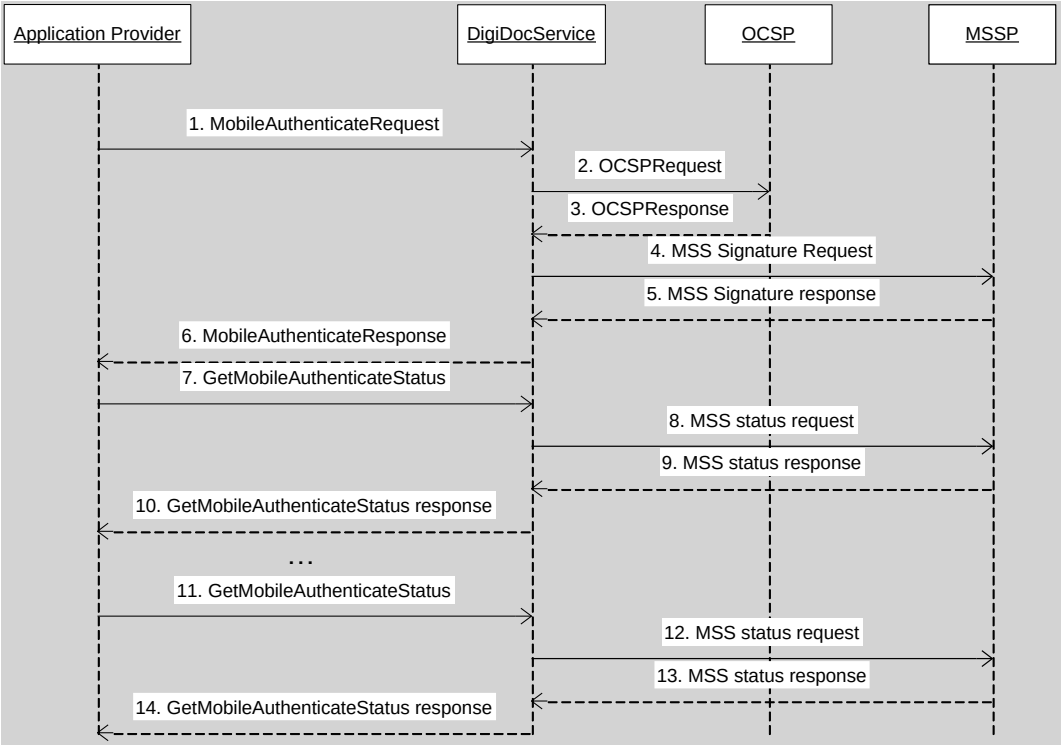
3. SessionCode is returned to the application, what enables the following procedures with the session data.
4. Before signing the application may add supplementary data files (AddDataFile request) or remove some datafiles (RemoveDataFile request) or carry out some other procedures with session data.
5. After procedures the current session document information is returned.
6. 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 located on the webpage reads the signer's certificate information from the smartcard. It is recommended to use Javascript library idCard.js for loading signature modules - available from www.id.ee
7. The certificate from the signer's smartcard together with other user inserted signature attributes is forwarded to the signing web-server.
8. Signature parameters are forwarded to DigiDocService with PrepareSignature request.
9. DigiDocService adds new signature information to the session document – signer's certificate and signature parameters and calculates the hash, what should be signed by the signer. The signed hash is sent to the application provider in PrepareSignature response.
10. 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.
11. The signature is forwarded to the signing web-server (application provider).
12. The signature is forwarded to DigiDocService with FinalizeSignature request.
13. DigiSocService makes a validity confirmation request about the validity of the signer's certificate to the OCSP validity service.
14. OCSP validity confirmation server returns the validity confirmation of the signature.
15. 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.
16. Application asks for the content of the DigiDoc file with GetSignedDoc request.
17. DigiDocService returns the current DigiDoc document which also contains the added signature.
18. 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.

19. 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 Client-Server mode



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 GetMobileAuthenticateStatus 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. 12. 13. 14 etc - this loop (7. 8. 9. 10.) goes on until positive answer or error message will arrive.

6.3.2 Authentication using smartcard

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

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. It is recommended to use both input parameters IDCode and PhoneNo! In case of Lithuanian Mobile-ID both IDCode and PhoneNo are mandatory.
CountryCode	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).. 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. It is recommended to use both input parameters IDCode and PhoneNo! In case of Lithuanian Mobile-ID users IDCode and PhoneNo are BOTH mandatory. (see chapter 5.2). If the element "PhoneNo" has been set, the country attribute set in the prefix is used (independent on the value of the element "CountryCode").
Language	String(3)	+	Language for user dialog in mobile phone. 3-letters

			capitalized acronyms are used. Possible values: EST, ENG, RUS, LIT
ServiceName	String(20)	+	Name of the service – previously agreed with Application Provider and DigiDocService operator. Maximum length – 20 chars.
MessageToDisplay	String(40 bytes)	-	Text displayed in addition to ServiceName and before asking authentication PIN. Maximum length is 40 bytes. In case of Latin letters, this means also a 40 character long text, but Cyrillic characters may be encoded by two bytes and you will not be able to send more than 20 symbols.
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 is recommended to always fill this field with a different random value every time. When authentication succeeds, it is recommended to verify that the user signed a message that contains this challenge value. (For more information about signature verification, see the description of the "Signature" element for the "GetMobileAuthenticateStatus" operation.)
MessagingMode	String	+	Mode to be used to respond to the 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.
ReturnRevocation Data	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 occurs, a SOAP error object is returned. Description of the SOAP error object and list of error codes are described in section 9.4.
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 authenticity 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 shall start sending GetMobileAuthenticateStatus queries until error message or positive answer will be returned.

NB! 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. Mobile-ID transactions will time out in 4 minutes or less.

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 BASE64 encoding. Returned only if SPChallenge field in the query was set. For more info,

		see the “Signature” field in the chapter 7.2 and chapter 4.2

7.2 GetMobileAuthenticateStatus

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 a response from MSSP before responding. If “FALSE” then response is returned immediately and the application should invoke GetMobileAuthenticate again after a small delay (2-10 seconds).

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 receiving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - INTERNAL_ERROR – technical error.
Signature	String	- Raw signature value in Base64 encoding. Returned only if SPChallenge field in the query was set in the MobileAuthenticate request. - - NB! For security reasons it is recommended that application providers verify this signature. The signature signs the challenge value that was returned by the MobileAuthenticate call (the “Challenge” field). It should also be verified that the first 10 bytes of this challenge were chosen by the application provider (that is, they should be equal to the value “SPChallenge” that was passed into

		MobileAuthenticate). - - Note that the authentication signatures are calculated without hash functions, both in the case of RSA and ECDSA. For example, if the challenge was "12345678901234567890369330D3483DAED0496D", (where the first half was chosen by the application provider), then the algorithm proceeds as if this challenge was actually a hash value. Therefore, for RSA, the usual SHA-1 prefix is prepended to the challenge before signing (even though the value did not come from SHA-1; this is the standard SHA-1 prefix from PKCS #1). As usual, ECDSA does not have prefixes. - The signature should be verified against the authentication certificate of the given user (as returned by the "CertificateData" field in MobileAuthenticate).

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 (including ID-card and other smartcard certificates and also digital stamp certificates issued by AS Sertifitseerimiskeskus) and number of foreign Certification Authorities. Additional info is available from the sales department of Sertifitseerimiskeskus.

Additionally, this operation returns the values of the most important fields from the certificate.

Query:

Parameter	Type	R	Description
Certificate	String	+	Certificate to be checked for validity, in Base64 format. May include „---BEGIN CERTIFICATE---„ and „---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 organization, this value will be taken from certificate subject's O (Organization) 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

8.1 StartSession

In most cases the transaction with the service is started using the StartSession method. It possible to also send data files with this operation; such files will be stored in session and can be operated on later. More precisely, there are 3 different ways to use StartSession:

1) The request can contain a DigiDoc or BDOC container. This is useful for signing and verifying existing containers, for adding or removing data files from the container, and also for extracting data file contents. To use this option, use the "SigDocXML" parameter. (Conversely, the "datafile" parameter should be left empty.)
2) A session can also be started without any data files. This is useful for example for creating new BDOC containers (which can be accomplished by invoking the "CreateSignedDoc" operation next). In this case, both parameters should be empty: "SigDocXML" and "datafile".

3) There is also an option for creating DigiDoc containers directly from this operation (this option only works for DigiDoc containers; BDOC can be created with the “CreateSignedDoc” operation). To use this operation, “SigDocXML” parameter should be empty, “datafile” parameter should be filled.

In the course of the StartSession’s query a unique session identifier is returned, which should be added to every procedure called within the transaction.

Query:

Parameter	Type	R	Description
SigningProfile	String	-	This value is currently ignored and may be empty.
SigDocXML	String	-	BDOC or DDOC document. A DigiDoc in XML transformed to HTML-Escaped format. For example “<DataFile>” should be transformed to „<DataFile>“. The container in BDOC format should be coded to BASE64 before it is delivered to the service.
bHoldSession	Boolean	-	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. The default value is “false”.
datafile	Datafile	-	Given parameter enables to send to service a data file within the StartSession request. Based on the file a DigiDoc container is created. (The BDOC format is not supported in this use case – please see the “CreateSignedDoc” operation). For example, when sending a “cv.pdf”, a “cv.ddoc” is created which contains the “cv.pdf“ only. The structure of a datafile element is described in chapter 9.3. While adding the datafile it’s unnecessary to determine the identifier. By default, DIGIDOC-XML 1.3 format fis created.

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

Response:

Parameter	Type	Description
Status	String	Value „OK” or an error string
Sesscode	Integer	Session code used for further requests in the given transaction.
SignedDocInfo	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.

8.1.1 HASHCODE

Normally, a size limit of 4 MB applies to digitally signed containers and data files sent to DigiDocService. To use bigger files, a “HASHCODE” mode of operation is supported by DigiDocService, where only hashes of data files are sent to the server (in other words, file contents are not sent to the service). This can also improve performance for large data files, as sending bigger files over network can take time.

To use this HASHCODE mode, the DigiDoc or BDOC-container should be converted to the HASHCODE form before sending it to the service. (In this form, the data file contents are replaced with their hash values.) Similarly, when the container is returned by the service, it should be converted back to regular form by inserting back the data file contents. Thus, in the end, the application provider still has a regular container that can be verified by standard tools (i.e., the DigiDoc3 client software). The exact form of HASHCODE depends on the particular format used (DDOC or BDOC).

8.1.1.1 BDOC format and HASHCODE

8.1.1.1.1 Transforming a BDOC container to HASHCODE form

Given the BDOC format, the container should be transformed to the HASCODE form in the following steps (before sending it to the DigiDocService):

1) Remove **all signed** files from the container. As the BDOC container is a ZIP-file, this can be accomplished with standard ZIP-file tools. The signed files are in the root folder of the ZIP file.
2) Add hashes of the removed files into the container. Two hash files need to be added, to the following locations in the BDOC (ZIP) file:

   - `META-INF/hashcodes-sha256.xml`
   - `META-INF/hashcodes-sha512.xml`

   The first hash file contains the SHA-256 hashes of all of the signed files. Similarly, the second file contains SHA-512 hashes for the same files. Both files have the same format: for every signed file, a full path is listed (as it appears in the BDOC file), the hash of the file in Base64, and also the length of the file in bytes. (The XML-schema is below). Hash values are calculated directly on the file contents (i.e., not over XML elements, which is different from the HASHCODE form for DDOC).

   As an example, if the container has 2 documents, named “file1.txt” and “File2.docx” (the sample files are downloadable from http://www.id.ee/public/bdoc_hashcode_example.zip), the corresponding hash files have the following contents:

   - *META-INF/hashcodes-sha256.xml*:

     ```
     <?xml version="1.0" encoding="utf-8"?>
     ```

```
<hashcodes>
    <file-entry full-path="file1.txt"
hash="fo+6a5j64VcKWJwvXsJE8PlB3tAdQ8/uwHAL5AEWmbk=" size="189"/>
    <file-entry full-path="File2.docx"
hash="3v5ZupBhiNxkCmmVKbtwwJKVCKxTZrQDPpNKF02ZiPo=" size="11665"/>
</hashcodes>
```

- *META-INF/hashcodes-sha512.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<hashcodes>
    <file-entry full-path="file1.txt"
hash="WIJZPgHWMrqfHqH7Arfjo8ymMZvI0IUgG8G8UESbnHXcpEPgOKutPph1GYOcSprj08VZ
a0m+myhlVPH29ThjIA==" size="189"/>
    <file-entry full-path="File2.docx"
hash="3z7gxofgCPoX2feWB9TQhUIvOlhsxm9RVR3iEFcCZ7uPcZuRc+KS9evmBC6bAMUnQOvk
ygXNTPfTIKb50krYYg==" size="11665"/>
</hashcodes>
```

The hash files must conform to the following XML-schema:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="hashcodes" type="hashcodesType"/>

  <xs:complexType name="fileEntryType">
    <xs:attribute name="full-path" type="xs:string" use="required"/>
    <xs:attribute name="hash" type="xs:string" use="required"/>
    <xs:attribute name="size" type="xs:long" use="required"/>
  </xs:complexType>

  <xs:complexType name="hashcodesType">
    <xs:sequence>
      <xs:element name="file-entry" type="fileEntryType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Note! Although the above ZIP operations can be done with any standard archive tools, some care must be taken to prevent accidental alterations to the file structure. Specifically, the ASiC standard mandates the following about the “mimetype” file:

- The “mimetype” file should (continue to) be the first file in the ZIP archive.
- The “mimetype” file should remain *not* compressed (it should be *stored*, not for example *deflated*).

After these steps, the HASHCODE form is ready. The contents of the BDOC file will then be analogous to the following diagram (note the addition of 2 hash files and absent data files):

```
├── META-INF
│   ├── hashcodes-sha256.xml
│   ├── hashcodes-sha512.xml
│   ├── manifest.xml
│   └── signatures0.xml
└── mimetype
```

The files in this example are downloadable (with BDOC containers) from the following address: http://www.id.ee/public/bdoc_hashcode_example.zip

8.1.1.1.2 Transforming the BDOC container back to standard form

When the HASHCODE container is returned back by DigiDocService, analogous steps need to be carried out:

1) The data files need to be added back into the container (the container is in ZIP-format, as all BDOC files are in ZIP-format). Note the following:

   - The data files in the container need to have a ZIP-file comment about the BDOC-library. In practice, this comment may simply be copied from other files in the archive (for example, from the „mimetype“-file). Note that the signature files (META-INF/signatureN.xml) should keep their existing comments (this is to preserve information about the tools used for signing particular signatures). An example of such comment is the following:
     `LIB DigiDocService/3.6.4 format: BDOC/2.1 Java: 1.7.0_51/Oracle Corporation OS: Windows 8/amd64/6.2 JVM: Java HotSpot(TM) 64-Bit Server VM/Oracle Corporation/24.51-b03`

   - Some care must be taken to prevent accidental alterations to the file structure. Specifically, the ASiC standard mandates the following about the “mimetype” file:
     - The “mimetype” file should (continue to) be the *first* file in the ZIP archive.
     - The “mimetype” file should remain *not* compressed (it should be *stored*, not for example *deflated*).

2) Remove all the hash files `hashcodes-*.xml` from the folder `META-INF`.

After these steps, the BDOC container is again in its normal form, and is ready to be used by for example the DigiDoc3 client software. Its contents should look similar to this:

```
├── META-INF
│   ├── manifest.xml
│   └── signatures0.xml
├── file1.txt
├── File2.docx
└── mimetype
```

8.1.1.2 DDOC format and HASHCODE

Example 1– 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**[1] form, where value „VGhpcyBpcyBhIHRlc3QgZmlsZQ0Kc2Vjb25kbGluZQ0KdGhpcmRsaW5l“ is previus datafile Base64 encoded and where there is added one newline before </DataFile> ending tag:

<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 canonization 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:

[1] Information about canonic XML: http://www.w3.org/TR/xml-c14n

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.

Example 2 – 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. Otherwise, it is not a proper DigiDoc format 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.

Query:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.

Response:

Parameter	Type	Description
Status	String	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.

Query:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
Format	String	+	a format of a document container to be created (currently supported formats are DIGIDOC-XML 1.3 and BDOC 2.1)
Version	String	+	a version number of the format of a creatable document container (currently the supported versions for DIGIDOC-XML is 1.3 and BDOC 2.1)

NB! Only container formats DIGIDOC-XML 1.3 and BDOC 2.0 are supported. If an inappropriate combination of given format and version number is used in request parameters, a SOAP error object with error message "Invalid format and version combination!" will be returned.

The description of DigiDoc formats are available on the webpage http://www.id.ee/index.php?id=36108 .

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK".
SignedDocInfo	String	SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

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. The size limit of 4 MB applies for DigiDoc containers and datafiles sent to Service. For bigger files content type HASHCODE could be used. See description below.

NB! Adding a data file is possible in the DigiDoc file with no signatures only.

Query:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
Filename	String	+	Name of the data file without the path.
MimeType	String	+	Type of the datafile.
ContentType	String	+	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*. Please see section 8.1. how to calculate hash from the source data file and how to send it to the service. **EMBEDDED_BASE64 –** The content of the file is in Base64 encoding in Content parameter.
Size	Integer	+	The actual size of data file in bytes.
DigestType	String	-	Hash code type of the data file. In case of DIGIDOC-XML format, "sha1" is supported; in case of BDOC, "sha256" is supported. Required in case of HASHCODE content type of file only.
DigestValue	String	-	The value of data file's hash in Base64 encoding.. Required for HASHCODE content type only. In case of the DIGIDOC-XML format, the hash is calculated over a DigiDoc <Datafile> element, using a canonicalized form (for more information, see chapter 8.1). In case of BDOC, the has is calculated over the binary data file content.
Attributes	String	-	Arbitrary amount of other attributes (meta data), what's add to <Datafile> element in DigiDoc file as attributes (in form <name>="<value>").
Content	String	-	The content of data file in Base64 encoding, is set if ContentType is EMBEDDED_BASE64.

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK”.
SignedDocInfo	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.

Query:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session
SignerIDCode	String	+	Identification number of the signer (personal national ID number). It is recommended to use both input parameters IDCode and PhoneNo! In case of Lithuanian Mobile-ID users SignerIDCode and SignerPhoneNo are mandatory.
SignersCountry	String	-	Country which issued the personal national ID number in ISO 3166-style 2-character format (e.g. “EE”)
SignerPhoneNo	String	+	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. It is recommended to use both input parameters IDCode and PhoneNo! In case of Lithuanian Mobile-ID users SignerIDCode and SignerPhoneNo are mandatory (see chapter 5.2) . If the element "SignerPhoneNo" has been set, the country attribute set in the prefix is used (independent on the value of the element "SignersCountry").
ServiceName	String	+	Name of the service – previously agreed with Application Provider and DigiDocService operator. Required, maximum length – 20 chars.
AdditionalDataToBeDisplayed	String	-	Additional text shown to the signer. Optional. Maximum length is 40 bytes. In case of Latin

			letters, this means also a 40 character long text, but Cyrillic characters may be encoded by two bytes and you will not be able to send more than 20 symbols.
Language	String	+	Language of the message displayed to the signer's phone. ISO 639 a 3-character-code in uppercase is used (for example EST, ENG, RUS, LIT).
Role	String	-	The text of the role or resolution defined by the user. Optional.
City	String	-	Name of the city, where it's signed. Optional.
StateOrProvince	String	-	Name of the state/province, where it's signed. Optional.
PostalCode	String	-	Postal code of the signing location. Optional.
CountryName	String	-	Name of the country, where it's signed. Optional.
SigningProfile	String	-	- "LT_TM" (Long Term with Time Mark): a profile for BDOC-TM (a BDOC signature with time-mark) and DDOC. "LT_TM" is currently the default option. - "LT" (Long Term): Used for creating standard BDOC-TS (BDOC with time-stamp / ASiC-E) signatures. Currently it is a reserved value that simply returns the error code 101 with the following message: "BDOC-TS signature format is not supported in the current service version. For signing BDOC files with Mobile-ID, please use BDOC-TM format". Support for the "LT" profile is planned for the service version 3.9.
MessagingMode	String	+	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.
AsyncConfiguration	Integer	-	Determines configuration used in asynchServerServer messaging mode. This shall be agreed previously between Application Provider and DigiDocService provider.
ReturnDocInfo	Boolean	+	If the value is true, the DigiDoc file information is returned as a result of the request.
ReturnDocData	Boolean	+	If the value is true, a DigiDoc document in HTMLescaped format SignedDocData element is returned.

Response:

Parameter	Type	Description
Status	String	"OK" or error message.
StatusCode	String	If the request is successful, 0 is returned, otherwise an error code.
ChallengeID	String	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 authenticity of the signing request.

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

NB! It is reasonable to wait at least 10 seconds before starting sending status queries - it is improbable that message from user's phone arrives earlier because of technical and human limitations. Mobile-ID transactions will time out in 4 minutes or less.

In case asynchServerServer messaging mode is used, a message will be sent from DigiDocService according 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 as HTML encoded if ReturnDocInfo was set "false" and ReturnDocData was set "true" in the request. c) Empty if both 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 received or error condition is detected. If the value is “false”, the response is returned immediately and the GetStatusInfo invocation should be repeated after a short time interval (2-10 seconds).

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 receiving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - REVOKED CERTIFICATE – certificate status revoked; - INTERNAL_ERROR – technical error,
StatusCode	String	Status code of the last request. In case of successful request, "OK" or an error string.
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:

Parameter	Type	R	Description

Sesscode	Integer	+	An identifier of the active session

Response:

Parameter	Type	Description
Status	String	"OK" or an error message
SignedDocInfo	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 receive the document information in structured format in addition to signed document, the GetSignedDocInfo request should be used.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session

Response:

Parameter	Type	Description
Status	String	"OK" or an error message
SignedDocInfo	SignedDocInfo	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 file system or to database.

8.9 GetDataFile

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.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session
DataFileId	String	+	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.

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK”.
DataFileData	DataFileInfo	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!”.

8.10 RemoveDataFile

RemoveDataFile request is for removing datafile from DigiDoc container. NB! Removing datafile is allowed when container to not have any signature. If container has one or more signatures, removing datafile is not possible.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
DataFileId	String	+	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.

Response:Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK”.
SignedDocInfo	SignedDocInfo	The document in the session info after removing the datafile. SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

If removing the datafile is unsuccessful, a SOAP error-object will be returned with an error-message. Ie when you try to remove datafile from signed document error “Cannot change a signed doc” is returned.

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.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
SignatureId	String	+	A 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.

Response:Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK".
SignedDocInfo	SignedDocInfo	The document in the session info after removing the signature. 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).

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
SignatureId	String	+	A 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.

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK”.
CertificateData	String	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).

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
SignatureId	String	+	A 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.

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK”.
CertificateData	String	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.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
SignatureId	String	+	A 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.

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK".
OcspData	String	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.

Response:

Parameter	Type	Description
Name	String	Name of the service (currently DigiDocService).
Version	String	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	String	DigiDoc library name
Libver	String	DigiDoc library 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 signing module of the user's computer.

Request:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
SignersCertificate	String	+	signer's certificate transferred to HEX string format (from binary (DER) format). Mostly the signing software (signing component) in the user's computer delivers the certificate in a proper format.
SignersTokenId	String	+	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	String	-	The text of the role or resolution defined by the user.
City	String	-	Name of the city, where it's signed
State	String	-	Name of the state, where it's signed.
PostalCode	String	-	Postal code of the signing location.
Country	String	-	Name of the country, where it's signed.
SigningProfile	String	-	- "LT_TM" (Long Term with Time Mark): a profile for BDOC-TM (a BDOC signature with time-mark) and DDOC. "LT_TM" is currently the default option. - "LT" (Long Term): Used for creating standard BDOC-TS (BDOC with time-stamp / ASiC-E) signatures. Currently it is a reserved value that simply returns the error code 101 with the following message: "BDOC-TS signature format is not supported in the current service version. For signing BDOC files with Mobile-ID, please use BDOC-TM

			format". Support for the "LT" profile is planned for the service version 3.9.

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:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK".
SignatureId	String	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.). The identifier could be used later to remove the signature or request for signature attributes (signer's certificate, OCSP certificate, OCSP validity confirmation).
SignedInfoDigest	String	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:

Parameter	Type	R	Description
Sesscode	Integer	+	An identifier of the active session.
SignatureId	String	+	The unique identifier of the signature which was returned as the result of PrepareSignature method
SignatureValue	String	+	Value of the signature (signed hash) as a HEX string. The signing software returns the value.

Response:

Parameter	Type	Description
Status	String	If the request is successful, the value will be „OK".
SignedDocInfo	SignedDocInfo	The document in the session info after adding the signature. SignedDocInfo structure will be returned in the format demonstrated in chapter 9.1.

8.18 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 and 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.

NB! Container formats DIGIDOC-XML 1.3 and BDOC 2.1 are supported. If an inappropriate combination of given format and version number is used in request parameters, a SOAP error object with error message "**Invalid format and version combination!**" will be returned.

Request:

Parameter	Type	R	Description
IDCode	String	+	Personal Identification Code of the user It is recommended to use both input parameters IDCode and PhoneNo! In case of Lithuanian Mobile-ID users IDCode and PhoneNo are mandatory.
SignersCountry	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). 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. It is recommended to use both input parameters IDCode and PhoneNo! In case of Lithuanian Mobile-ID users IDCode and PhoneNo are mandatory (see chapter 5.2). If the element "PhoneNo" has been set, the country attribute set in the prefix is used (independent on the value of the element "SignersCountry").
Language	String(3)	+	Language for user dialog in mobile phone. 3-character capitalized acronyms are used. Possible values: ENG, EST, RUS. LIT.
ServiceName	String(20)	+	Name of the service – previously agreed with Application Provider and DigiDocService operator. Maximum length – 20 chars.
MessageToDisplay	String(40 bytes)	-	Text displayed in addition to ServiceName and before asking authentication PIN. Maximum length is 40 bytes. In case of Latin letters, this means also a 40 character long text, but Cyrillic characters may be encoded by two bytes and you will not be able to send more than 20 symbols.

<table>
<tr><td>Role</td><td>String</td><td>-</td><td>Role or resolution of the signature</td></tr>
<tr><td>City</td><td>String</td><td>-</td><td>City where the signature is created</td></tr>
<tr><td>StateOrProvince</td><td>String</td><td>-</td><td>State or province where the signature is created</td></tr>
<tr><td>PostalCode</td><td>String</td><td>-</td><td>Postal code of the place where the signature is created</td></tr>
<tr><td>CountryName</td><td>String</td><td>-</td><td>Country where the signature is created</td></tr>
<tr><td>SigningProfile</td><td>String</td><td>-</td><td>- “LT_TM" (Long Term with Time Mark): a profile for BDOC-TM (a BDOC signature with time-mark) and DDOC. “LT_TM” is currently the default option.
- “LT” (Long Term): Used for creating standard BDOC-TS signatures (BDOC with time-stamp / ASiC-E); it is supported for the BDOC container format.</td></tr>
<tr><td>Datafiles</td><td>List</td><td>+</td><td>List of the files to be signed. Every element has following fields:
<table>
<tr><td>Parameter</td><td>R</td><td>Description</td></tr>
<tr><td>Id</td><td>+</td><td>unique identifier for the file. In case of DIGIDOC-XML format, the identifiers of the data files start with „D“ followed by a sequential number of the file. In case of BDOC format a unique file name is transferred.</td></tr>
<tr><td>DigestType</td><td>+</td><td>hash algorithm identifier. In case of DIGIDOC-XML format the supported type is "sha1". In cased of BDOC format, the recommended type is "sha256".</td></tr>
<tr><td>DigestValue</td><td>+</td><td>hash value of the data file in BASE64 encoding. In case of DIGIDOC-XML format, hash is calculated over DigiDoc <Datafile> element canonic form. Please see section 8.1 how to calculate hash over data file and send it to the service. For the BDOC form, hash is calculated over the binary datafile contents and then is encoded in Base64.</td></tr>
<tr><td>MimeType</td><td>-</td><td>Type of the data file. In case of BDOC, the default value is "application/octet-stream". Note! In case of BDOC, it is very important that in the container the manifest.xml file contains the same MimeType for this file.</td></tr>
</table></td></tr>
<tr><td>Format</td><td>String</td><td>+</td><td>Format identifier for the signed file, shall equal to “DIGIDOC-XML” and "BDOC".</td></tr>
<tr><td>Version</td><td>String</td><td>+</td><td>Format version of the undersigned file (In case of</td></tr>
</table>

			DIGIDOC-XML, the supported version is "1.3", in case of BDOC, it is "2.1").
SignatureID	String	+	- Identifier of the signature. The Application Provider shall detect identifier of the latest signature and increment this value by one. For example, if last signature has ID of value “S2”, the value of this parameter should be “S3”. - In case the document has no signatures, the value should be “S0”.
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.
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.

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 9.4 of the current document.

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

NB! 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. Mobile-ID transactions will time out in 4 minutes or less.

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 following structure:

Parameter	Type	Description
Sesscode	Integer	Identifier of the current active session
Status	String	Status code. "SIGNATURE" in case of successful

		signing. Other possible status codes are described in GetMobileSignatureStatus responses.
Data	String	The resulting <Signature> block in pure XML.

8.19 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", the response is not returned to the request before the signature value has arrived from the phone or an error has occurred. If "False", the response will be returned immediately and the Application provider has to repeat the request after some time (preferably in 2-10 seconds).

Response:

Parameter	Type	Description
Sesscode	Integer	Session identifier
Status	String	Process status: - REQUEST_OK – the original message was successfully received; - EXPIRED_TRANSACTION – service timed out before user managed to complete the signing; - 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; - MID_NOT_READY – the Mobile-ID 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 receiving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - NOT_VALID - signature is not valid - REVOKED_CERTIFICATE – certificate revoked - INTERNAL_ERROR – technical error.
Signature	String	Signature value in PKCS#1 container in BASE64 encoding. Can be either an RSA or ECDSA signature, depending on the signer's certificate returned with the signature block.

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

8.20 GetMobileCertificate

The method is used to request user's certificates.

NB! The usage of this method is limited (IP-address based access). It is necessary to request the separate access from SK with clear argument why it is needed.

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). 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. If the element "PhoneNo" has been set, the country attribute set in the prefix is used (independent on the value of the element "Country").
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.

Response:

Parameter	Type	Description
AuthCertStatus	String	OK – the authentication certificate has not expired. Note that the certificate may still be inactive for other reasons (it may be revoked by its owner). REVOKED – certificate has expired. The application provider may additionally ask for definitive certificate status by using an OCSP service (for example using the "CheckCertificate" operation).
SignCertStatus	String	OK – the signing certificate has not expired. Note that the certificate may still be inactive for other reasons (it may be revoked by its owner). REVOKED – certificate has expired. The application provider may additionally ask for

		definitive certificate status by using an OCSP service (for example using the “CheckCertificate” operation).
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.

8.21 MobileSignHash

This operation starts the process of signing a hash using Mobile-ID. It is meant for signing document formats other than DDOC and BDOC (for example: PDF, ADOC, etc.) For BDOC and DDOC formats, it is recommended to use the MobileCreateSignature and MobileSign operations.

If it is necessary to fetch signer’s certificate before signing (for example, to incorporate the certificate in the document prior to signing), the GetMobileCertificate operation can be used.

This operation locates the signer’s certificate, fetches an OCSP response and sends the signing request to the signer’s mobile device. An active session is not required.

The status of the hash signing process is checked in ClientServer mode with the GetMobileSignHashStatusV2 operation. Note! Before sending the first status request, it is recommended to wait at least 10 seconds, as the signing process cannot finish faster due to human and technology factors. Mobile-ID transactions will time out in 4 minutes or less.

This operation is using the document/literal style and is accessible from a new sub-address /v2/?wsdl. New version of the service uses a separate WSDL, and error message format has been updated (see chapter 9.4).

NOTE: The usage of this method is limited (IP-address based access). It is necessary to request the separate access permissions for using it.

Request:

Parameter	Type	R	Description
IDCode	String	+	Personal Identification Code of the user
PhoneNo	String	+	Phone number of the certificate number complete with the country code in the form +xxxxxxxxx (e.g. +3706234566). A match between the phone number and the ID-code will be checked and in case on non-compliance a SOAP error code 301 will be returned,
Language	String(3)	+	Language of the messages displayed on user's phone. ISO 3166 3-letter codes are being used. Possible values are: EST, ENG, LIT and RUS.
MessageToDisplay	String(40)	-	Text displayed in addition to ServiceName and before asking authentication PIN. Maximum length is 40 bytes. In case of Latin letters, this means also a

			40 character long text, but Cyrillic characters may be encoded by two bytes and you will not be able to send more than 20 symbols.
ServiceName	String(20)	+	Name of the service – previously agreed with Application Provider and DigiDocService operator. Maximum length – 20 chars.
Hash	String(128)	+	A hash to be signed. 128 characters. Transferred as a HEX string.
HashType	String(20)	+	A hash type to be signed. Sha1, sha256 and sha512 hashes are currently supported.

Response:

Parameter	Type	Description
Sesscode	String	Identifier of the session.
ChallengeID	String	- 4 (number) character control code, which is calculated on a basis of Challenge value that will be sent to the user's phone for signing. - 40 characters long HEX i.e. hash Challenge to be signed. Will be used only in case of Bite MSSP operator. This control code shall be displayed to the user by the application; with this it will be possible for the user to prove the authenticity of the request. NOTE: Application must prompt the user to check the compatibility of the control code displayed in the application and on the phone screen.
Status	String	"OK" if the procedure was performed successfully. If method call-up will result with an error, a SOAP error object will be returned.

If method call-up will result with an error, a SOAP error object will be returned according to the description in chapter 9.4.

8.22 GetMobileSignHashStatusRequest

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

This operation returns the status of the hash signing operation and, in the case of successful signing, the signature, signer's certificate and revocation data about the certificate.

Request:

Parameter	Type	R	Description
Sesscode	String(20)	+	Identifier of the session.
WaitSignature	Boolean	-	If "True", the response is not returned to the request before the signature value has arrived from the phone or an error has occurred. If "False", the response will be returned immediately and the

			Application provider has to repeat the request after some time (preferably in 2-10 seconds).

Response:

Parameter	Type	Description
Sesscode	String	Identifier of the session.
Status	String	Process status: - OUTSTANDING_TRANSACTION – authentication is still on the way, the status - SIGNATURE – signature was successfully created; - NOT_VALID – the action is completed but the signature created is not valid; - EXPIRED_TRANSACTION – service timed out before user managed to complete the signing; - USER_CANCEL – user cancelled the action; - query shall be repeated; - MID_NOT_READY – the Mobile-ID 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 receiving the message, error in messaging server etc.); - SIM_ERROR – SIM application error; - INTERNAL_ERROR – technical error - REVOKED_CERTIFICATE – certificate revoked or suspended - OCSP UNAUTHORIZED - the client who is using the service does not have access to validity confirmation service of OCSP used by DigiDocService.
Signature	String	Signed hash in a PKC1 / PKCS13 container. (Will be returned only if Status == "SIGNATURE"). Can be either a RSA or an ECDSA signature, depending on the returned certificate in the “CertificateData” field.
RevocationData	String	Validity information of the certificate (PEM format)
CertificateData	String	Certificate in PEM format, encoded in Base64.

If method call-up will result with an error, a SOAP error object will be returned according to the description in chapter 9.4.

9 Data structures

9.1 SignedDocInfo

Presents the structure of a DigiDoc file (container).

- **Format** – File format for the signed container (DIGIDOC-XML and BDOC are supported currently).

- **Version** - The version of a signed file format (in case of DIGIDOC-XML the versions 1.1, 1.2, 1.3; in case of BDOC the version 2.1).
- **DataFileInfo –** Information about the files in container. The data structure is described in chapter 9.3 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' identifiers 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. If the signature is valid, but doesn't completely correspond to the container's specification, the value of this element is "OK", while the Error-element has a description of the warning returned by the DigiDoc library.
    - **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:
          TECHNICAL – technical issue;
          USER – issue caused by user;
          LIBRARY – internal error of the DigiDoc library.
          WARNING – A warning from the DigiDoc library. Legally, the signature is valid, but additional changes are not allowed in the container. For more information, see chapter 9.5.
        - **description –** Error description in English.
    - **SigningTime** – Local time (for example, 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** - Information about the RFC3161 timestamps that are related to the signature. The timestamps functionality is not realized in the service version
  - **CRLInfo** - Information about signature related revocation list. The revocation list related functionality is not realized in the service version

Sample of structure:

```
<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:DataFileAttribute">
         <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 primly).
  - **Description –** A short description of signing policies.

Sample of structure:

```
<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/seria
lNumber=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** – unique identifier of a file. In case of DIGIDOC-XML format, the data file identifiers start with a symbol „D" followed by the file's sequence number. In case of BDOC format the identifier is the file name, which must be unique. 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 datafile.
- **Size** – The actual size of file in bytes.
- **DigestType** - Hashcode type of the data file. In case of DIGIDOC-XML format the form currently supported algorithm is "sha1", in case of BDOC format the supported algorithm is "sha256". 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.

A new structure of error objects is being used in the responses of the methods MobileSighHash and GetMobileSignHashStatus. The element <faultstring> contains the error code. A subelement of the element <detail> is <endpointError> type of object that contains one <message> element with a message that explains the error message, and a zero or more <reason> elements with detailed descriptions of the errror (please see the examples at the end of the chapter).

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 are missing
103	Service provider does not have access to SK validity confirmation service (OCSP response UNAUTHORIZED)
200	General error of the service
201	Missing user certificate
202	Unable to verify certificate validity
203	Session is locked by the other SOAP request.
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)
304	Certificates is suspended
305	Certificate is expired
413	Incoming message exceeds permitted volume limit.
503	The number of simultaneous requests of the service has been exceeded.

Example 1 of the service error message:

In the request (the first version service, old structure) the phone number format was incorrect or the country code of the phone number was not included in the list of supported country codes.

```
<SOAP-ENV:Fault>
   <faultcode>SOAP-ENV:Client</faultcode>
   <faultstring xml:lang="en">102</faultstring>
   <detail>
      <message>User IDcode and Phone number are
mandatory</message>
   </detail>
</SOAP-ENV:Fault>
```

Example 2 of the service error message:

Several errors were identified in the request (the second version service, new structure).

- Sequence of the request parameters was incorrect. Parameter "IDCode" was expected as the first one.
- Phone number does not correspond to the expected type

```
<SOAP-ENV:Fault>
   <faultcode>SOAP-ENV:Client</faultcode>
   <faultstring xml:lang="en">101</faultstring>
   <detail>
      <endpointError>
         <message>Request message validation
failed</message>
```

```
         <reason>cvc-complex-type.2.4.a: Invalid content was
found starting with element 'MessageToDisplay'. One of
'{IDCode}' is expected.</reason>
         <reason>cvc-minLength-valid: Value '' with length =
'0' is not facet-valid with respect to minLength '5' for
type 'PhoneNumberType'.</reason>
         <reason>cvc-type.3.1.3: The value '' of element
'PhoneNo' is not valid.</reason>
      </endpointError>
   </detail>
</SOAP-ENV:Fault>
```

9.5 Container validation

When an existing container is sent to DigiDocService, the document is automatically validated. Starting from JDigiDoc library version 3.8 certain technical errors are allowed in the container for compatibility reasons. Such errors are reported as warnings.

When a container has warnings, the signatures are considered legally valid (and thus the status of these signatures is “OK”, even though an “Error” element is also present). For some warnings, further modifications to the document are not allowed (adding and removing signatures is not permitted). The following errors are considered warnings:

- 129 WARN_WEAK_DIGEST – the container uses a weak hash algorithm (e.g., SHA-1). Adding new signatures is not allowed.
- 173 ERR_DF_INV_HASH_GOOD_ALT_HASH – the XML DataFile element is missing a namespace attribute. Adding signatures is not allowed.
- 176 ERR_ISSUER_XMLNS – The XML elements X509IssuerName and/or X509SerialNumber are lacking namespace attributes.
- 177 ERR_OLD_VER – The container version is not supported anymore. Adding signatures is not allowed.

For more information about warnings, please refer to the JDigiDoc library documentation: http://www.id.ee/public/SK-JDD-PRG-GUIDE.pdf , chapter “Validation status VALID WITH WARNINGS”.

The warnings are only visible in responses under the SignatureInfo elements. The Status element under SignatureInfo has the value “OK”, but there is also an “Error” element present, whose category is set to “WARNING”. The warning is specified in the “Code” and “Description” elements.

10 Service Change History

Service changes' history can be found at http://www.id.ee/?lang=en&id=36458