DigiDoc C library

Document version: 2.2.5, 27.03. 2006

Contents:

This document describes C library of DigiDoc system. The library is used by number of digital signature applications and is a basic building tool for building future applications for handling digital signatures.

COM library of DigiDoc system is described in separate document.

References and abbrevations

RFC2560	Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C., X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP. June 1999.
RFC3275	Eastlake 3rd D., Reagle J., Solo D., (Extensible Markup Language) XML Signature Syntax and Processing. (XML-DSIG) March 2002.
ETSI TS 101 903	XML Advanced Electronic Signatures (XAdES). February 2002.
XML Schema 2	XML Schema Part 2: Datatypes. W3C Recommendation 02 May 2001. http://www.w3.org/TR/xmlschema-2/
DAS	Estonian Digital Signature Law
ESTEID	Estonian ID-card
CSP (MS CSP)	Microsoft Crypto Service Provider
PKCS#11	RSA Laboratories Cryptographic Token Interface Standard http://www.rsasecurity.com/rsalabs/PKCS/index.html

Usage of DigiDoc C library

Overview

Introduction

DigiDoc C library is designed for handling of digital signatures covering all necessary functions like creation and verification of digitally signed files. DigiDoc C library produces digitally signed files in “DigiDoc

format" compliant to XML-DSIG and XadES standards. The file format is described in separate document.

DigiDoc C library is dependant on following base libraries:

- **OpenSSL** – version 0.9.7 or newer. URL to this library is ftp://ftp.openssl.org/snapshot/
- **libxml2** – version 2.5.1 or newer. URL-s to source code are ftp://ftp.gnome.org/pub/GNOME/stable/sources/libxml/ and http://www.xmlsoft.org/. Pre-compiled binaries are available from: http://www.zlatkovic.com/projects/libxml/index.html.

Smartcard, smartcard reader and appropriate drivers are required for creation of digitally signed files. The C library supports signing with smartcard over Microsoft Crypto API and PKCS#11 interfaces.

DigiDoc C library provides methods for creation and handling of various cryptographic data structures without binding them directly to some file format (XML vs. PKCS#7) or particular cryptographic library. A number of C data structures are defined for saving data, use of data structures from OpenSSL and libxml2 libraries is avoided.

C library files

The library has been successfully compiled under Linux (Fedora Core, Suse and Mandrake) and Windows (NT and 2000) operating systems. Smartcard and smartcard reader with appropriate drivers are required for some functions of the library.

DigiDoc C library consists of following files:

1. DigiDocLib.c – main source file containing most of the functions, including handling of digitally signed files is DigiDoc format, verification and validity confirmation handling functions.
   Some of signature creation functions make use of ESTEID card and its specifics and are therefore platform-dependant. Signing through PKCS#11 platform has been implemented. Signing thorough MS CSP drivers on Windows platformhas been implemented in a separate library – MS COM component.
   Some functions contain blocks of program code which is compiled in only in non-Windows platforms. For excluding Windows-specific code, avoid pre-processor constant WIN32.

2. DigiDocDefs.h – this file contains global definitions
3. DigiDocPKCS11. h/c – functions for signing through PKCS#11 driver

4. DigiDocConfig. h/c – functions for reading configuration file and also some simple functions for signing and verification of signatures and validity confirmations.
5. DigiDocError. h/c – error handling routines and explanations.
6. DigiDocMem. h/c – Memory management functions
7. DigiDocStack. h/c – Simple stack of xml tags used in SAX parser.
8. DigiDocCert. h/c – certificat handling functions
9. DigiDocDebug. h/c – debugging functions
10. DigiDocParser. h/c – digidoc parser utilizing the XMLReader API of libxml2
11. DigiDocSAXParser. h/c – main digidoc document parser using SAX API of libxml2
12. DigiDocConvert. h/c – conversion functions
13. DigiDocEnc. h/c – main data structures and access functions for XML Encryption & Decryption (XML-ENC)
14. DigiDocEncGen. h/c – functions for generation of xml files in XML-ENC format.
15. DigiDocEncSAXParser. h/c - parser for xmlfiles in XML-ENC format using SAX API of libxml2.
16. cdigidoc.c - sample command-line utility.

Components of library

The DigiDoc C library consists of three kinds of components:

- Data structures – Some of the data structures reflect tags and structures of DigiDoc file format, some are just for other puropses like `ErrorInfo` or `CertSearch`. Data structures are described later in this document. Declarations of data structures are found in file DigiDocLib.h.
- Constants – a number of constants is used by the library, including error codes. Constants are described later in this document, their definitions are found in files DigiDocLib.h and DigiDocError.h.
- Functions – defined in *.c files of the library. Functions of public interest are declared in fuile DigiDocLib.c. These are categorized as:
    - **General and administrative functions** – functions for the library itseld and also some conversion routines;
    - **General cryptographic functions** – hash value calculation, some general signing routines and some conversion functions making use of the OpenSSL library;
    - **General XML-producing functions** – those whose are not very DigiDoc format specific;

- **Management of signed document** – the biggest class of functions including using and adding structures, information retrieval from structures and signing functions;
- **Reading and writing of signed document** – basic functions for writing and reading DigiDoc file format;
- **Certificate functions** – this class consists of functions for searching, retrieving and parsing certificates;
- **Verification of signed document** - most of the "verify*" functions are here dealing with verification of different structures of DigiDoc file format.
- **Error handling functions**

- **XML Encryption & Decryption**

Basic usage examples

How to create a new DigiDoc format file?

The following is a brief overview how to create a DigiDoc format file using C library functions.

First, we define required structures:

```
SignedDoc* pSigDoc;
```

This structure reflects the file format of DigiDoc. All other relevant structures are part of this basic structure.

```
DataFile* pDataFile;
```

One DataFile structure corresponds to the original file (file-to-be-signed) in DigiDoc container. One DigiDoc container can incorporate multiple original files. The original file can be embedded in the DigiDoc file or it could be outside (detached).

Then we initialise the library:

```
initDigiDocLib();
```

This ensures all OpenSSL library parameters are properly initialised. Now we create DigiDoc structure in the memory.

```
rc = SignedDoc_new(&pSigDoc, DIGIDOC_XML_1_NAME,
DIGIDOC_XML_1_3_VER);
```

Values of these constant above are defined as "DIGIDOC-XML" and "1.3" (DigiDocLib.h). The library supports document versions 1.0, 1.1, 1.2 and

1.3. The 1.3 format shall be used as it corresponds properly to XAdES standard which has been proven in several international interoperability tests.

Data file(s) can be added after cration of DigiDoc structure:

```
rc = DataFile_new(&pDataFile, pSigDoc, NULL, infile,
CONTENT_EMBEDDED_BASE64, mime, 0, NULL, 0, NULL, CHARSET_UTF_8,
CHARSET_UTF_8);
```

Second parameter here is a pointer to DigiDoc structure where the data file (in the first parameter) is added.

Third parameter is an unique identificator for the document (const char*). If it is NULL then the library takes care of generation of it.

Fourth parameter is a name of the original file. It is recommended to include full path in this parameter; the path is removed when writing it to DigiDoc container file.

Fifth parameter reflects how original files are embedded in the DigiDoc container. Possible options are (DigiDocLib.h):

1. CONTENT_DETATCHED
2. CONTENT_EMBEDDED
3. CONTENT_EMBEDDED_BASE64

The first means that original file is not embedded in DigiDoc container. Instead only link to external file is embedded. This might be usable in case of large original files.

CONTENT_EMBEDDED copies contents of the original file into DigiDoc XML-container. This is dangerous option because of possibility of various extra symbols in the original file. It is therefore recommended to use CONTENT_EMBEDDED_BASE64 option instead. This encodes contents of the original file using base64-encoding before merging it into DigiDoc container.

Next parameter is a MIME type of the original file like "application-x/ms-word” or "application-x/acrobat-pdf" depending on the type of original file.

In most cases next four parameters shall be left to the library to calculate. These are:

1. Size of the original file in bytes
2. Hash of the original file
3. Size of the hash of the original file
4. Type of hash algorithm (only sha1 is supported)

Last two parameters deal with encodings of parameters. The former determines encoding of parameters in DigiDoc container, the latter determines encoding of file name. Two constants are defined for these parameters:

- CHARSET_ISO_8859_1
- CHARSET_UTF_8

In Windows environment it is required that file name is in ISO-8859-1 encoding.

This concludes description of function Datafile_new() parameters. After this we have saved original file data (and file itself) in DigiDoc container. We still miss values of these parameters we wanted library to calculate (see above). This is done by the following:

```
rc = calculateDataFileSizeAndDigest(pSigDoc, pDataFile->szId, infile,
DIGEST_SHA1);
```

This function calculates and adds these four values to section `pDataFile->szId` based on file name given in third parameter. DIGEST_SHA1 is the only supported hash algorithm. The function returns ERR_OK in case of success or error code.

It is possible to use function

```
char * w=getErrorString(rc);
```

to get error string in case of function returns with error code (rc != ERR_OK). Error strings are in english.

It is possible to use additional extra XMLatributes to the data file, Function `addDataFileAttribute()`is used for that. For example:

```
addDataFileAttribute(pDataFile, "ISBN", "000012345235623465");

addDataFileAttribute(pDataFile, "Owner", "CEO");
```

First parameter is pointer to original file structure followed by attribute name and value. Data shall be presented in UTF-8 encoding.

For creating file in DigiDoc format the following function is used:

```
rc = createSignedDoc(pSigDoc, oldfile, outfile);
```

Parameters to this function consist of DigiDoc structure to be saved and two file names. The “oldfile” parameter can be omitted and “outfile” is file name under which the container will be saved. If the “oldfile” is represented and it can be opened then contents of original files will be copied from there.

Memory shall be released after end of working with DigiDoc structure:

```
SignedDoc_free(pSigDoc);
```

This also releases memory used for keeping original files.

The last task is to shut down the library:

```
finalizeDigiDocLib();
```

How to add validity confirmation?

OCSP (*Online Certificate Status Protocol*) is used to get validity confirmation from OCSP Responder to prove that certificate was valid at the time of signing. Validity confirmation shall be retrieved immediately after creation of signature by user.

Function getConfirmation() creates OCSP request from given *pSigDoc* signature *pSigInfo*, according to function parameters signs the request (if it was required) and sends the request to address from parameter notaryURL.

Information about certificate in question and issuer of the certificate is also contained in OCSP request.

Response is included in DigiDoc container in *pSigDoc* using base-64 encoding. In case of success, ERR_OK is returned, otherwise error code is returned.

```
int getConfirmation(SignedDoc* pSigDoc, SignatureInfo* pSigInfo,

          const X509** caCerts, const X509* pNotCert,

          char* pkcs12FileName, char* pkcs12Password,

          char* notaryURL,

          char* proxyHost, char* proxyPort)
```

Parameters of the function:

- pSigDoc – structure of existing DigiDoc container
- pSigInfo – pointer to particular existing signature within DigiDoc container
- caCerts - chain of CA certificates from user certificate issuer up to self-signed root certificate
- pNotCert – pointer to OCSP Responder certificate
- pkcs12FileName – in case OCSP request needs to be signed, this parameter contains pointer to file name for private key/certificate container which shall be in PKCS#12 format. Use value NULL if request needs not to be signed.

- pkcs12Password – password for using PKCS#12 container from last parameter.
- notaryURL – URL for OCSP service
- proxyHost – proxy host name or NULL
- proxyPort – proxy port name or NULL

- pkcs12FileName – OCSP-päringu teostamiseks vajaliku pääsutõendi asukoht.

Example:

We will retrieve validity confirmation for file "c:\\tt.ddoc" signature "S0"

```
rc = readSignedDoc(&pSigDoc, "c:\\tt.ddoc", 1, buf);
pSigInfo = getSignatureWithId(pSigDoc, "S0");
rc = getConfirmation(pSigDoc, pSigInfo,
            caCerts, notSigCert,
            pkcs12FileName, pkcS12Password,
            "http://ocsp.sk.ee/", "proxy.intern.ee", "8080");
```

How to search for certificates?

Function from the last example needed three certificates: user's CA certificate, OSCP Responder certificate and certificate for signing the OCSP request. The latter is not required when OSCP Responder accepts unsigned requests.

DigiDoc C library can fetch certificates from three different sources: Microsoft CertStore, PKCS #12 container and regular PEM-encoded certificate file (hereinafter referred as "X509 file").

Structure `CertSearch_st` is used in certificate search functions, parameter `searchType` determines type of search.

```
typedef struct CertSearch_st {
      int searchType;
      char* x509FileName;
      char* keyFileName;
      char* pkcs12FileName;
      char * pswd;
      CertSearchStore* certSearchStore;
} CertSearch;
```

The parameter `searchType` can have three different values:

```
CERT_SEARCH_BY_STORE
CERT_SEARCH_BY_X509
CERT_SEARCH_BY_PKCS12
```

If CERT_SEARCH_BY_STORE is chosen then parameter certSearchStore shall have value, others can be omitted.

If CERT_SEARCH_BY_PKCS12 is chosen then parameters pkcs12FileName and pswd shall have value, others can be omitted.

If CERT_SEARCH_BY_X509 is chosenthen parameter x509FileName shall have value. If this certificate is used for signing (using software certificates) then additionally parameters keyFileName and pswd shall have value.

Structure CertSearchStore_st is used for searching certificates from Microsoft CertStore.

```
typedef struct CertSearchStore_st {
      int searchType;
      char* storeName; // default is "My"
      long certSerial;
      int numberOfSubDNCriterias;
      char** subDNCriterias;
      int numberOfIssDNCriterias;
      char** issDNCriterias;
} CertSearchStore;
```

Microsoft CertStore can be searched by three different parameters: by serial number, by subject DN (Distinguished Name) and by issuer DN. These methods can be combained. For example one can search certificate with certain serial number and subject name or with specific serial number and issuer name.

Constants for search methods are:

CERT_STORE_SEARCH_BY_SERIAL
CERT_STORE_SEARCH_BY_SUBJECT_DN
CERT_STORE_SEARCH_BY_ISSUER_DN
CERT_STORE_SEARCH_BY_KEY_INFO

These constans can me OR’d for searchType value.

If type of the search is by serial number then parameter certSerial shall have value. A little more complicated is to set parameters for subject and issuer.

In case searching by issuer, an array of strings shall be formed size of numberOfIssDNCriterias whereas strings go to issDNCriterias field. The same holds true when searching by subject only numberOfISubDNCriterias and issDNCriterias are involved. Strings in parameters are ot case sensitive and whitespaces are also removed before search.

As of example, lets find ESTEID-SK certificate (Estonian ID card CA) from Microsoft CertStore. It shall go as follows:

```
CertSearchStore certSearchStore;
certSearchStore.searchType=CERT_STORE_SEARCH_BY_SERIAL|
CERT_STORE_SEARCH_BY_ISSUER_DN;
certSearchStore.certSerial=0x3C445C82;
certSearchStore.issDNCriterias[0]=" CN = Juur-SK ";
certSearchStore.issDNCriterias[1]=" O = AS Sertifitseerimiskeskus ";
certSearchStore.numberOfIssDNCriterias=2;
certSearchStore.storeName="CA";


CertSearch certSearch;
certSearch.searchType=CERT_SEARCH_BY_STORE;
certSearch.certSearchStore=certSearchStore;
```

How to open and read DigiDoc files?

Let's have a look at sample code. Error-checking and less important declarations of variables are omitted for better readability.

```
SignedDoc* pSigDoc;
DataFile* dataFile;
SignatureInfo* pSigInfo;

initDigiDocLib();
rc = ddocSaxReadSignedDocFromFile(&pSigDoc, infile, 0, 0);
NumberOfDataFiles=getCountOfDataFiles(pSigDoc));
NumberOfSignatures=getCountOfSignatures(pSigDoc);
for(counter=0;counter<NumberOfDataFiles;counter++){
// NULL is returned in case there's no datafile
// with number "counter"
      dataFile = getDataFile(pSigDoc,counter);
      if(!dataFile){
            break;
      }
// Do something with dataFile
}
for(counter=0;counter< NumberOfSignatures;counter++){
      pSigInfo=getSignature(pSigDoc,counter);
      // Do something with signature info
}
SignedDoc_free(pSigDoc);
finalizeDigiDocLib();
```

Function `ddocSaxReadSignedDocFromFile()`is used for opening and reading DigiDoc files. Parameters to this function are:

- location of DigiDoc structure
- file name
- flag indicating whether to check hash value(s) of original file(s) is required at the time of opening

- maximum size of DataFile content to be cached in memory.

Other functions reveal contents of the DigiDoc file.

How to add signature to existing DigiDoc file?

Let's have a look at sample code. Error-checking and less important declarations of variables are omitted for better readability.

```
SignedDoc* pSigDoc;
SignatureInfo* pSigInfo;

initDigiDocLib();
rc = readSignedDoc(&pSigDoc, infile, 1, buf);

// add new signature with default id
rc = SignatureInfo_new(&pSigInfo, pSigDoc, NULL);
// automatically calculate doc-info elements for this signature
addAllDocInfos(pSigDoc, pSigInfo);
// add signature production place
setSignatureProductionPlace(pSigInfo, "Tallinn", "Harjumaa", "12345",
"Estonia");
// add user roles
addSignerRole(pSigInfo, 0, "VIP", -1, 0);

rc = calculateSigInfoSignatureWithCSPEstID(pSigDoc,pSigInfo);

rc = createSignedDoc(pSigDoc, infile, outfile);

SignedDoc_free(pSigDoc);
finalizeDigiDocLib();
```

First, the DigiDoc structure shall be created or fetched from file. In our example we read the structe from file using function `ddocSaxReadSignedDocFromFile()`.

Next we create a new SignatureInfo structure using function `SignatureInfo_new()`. The first parameter is a reference to the structure itself, the second one points out DigiDoc structure in question. The third parameter will uniquely identify the signature; it is recommended to submit NULL to this in order to let the library figure it out automatically.

Function `addAllDocInfos()` adds newly created SignatureInfo structure into the DigiDoc structure. Function `setSignatureProductionPlace()` allows to add user-defined signature production place into the structure. Function `addSignerRole()` adds role of signatory to the structure. This function has parameters as following:

- pointer to SignatureInfo structure

- whether the role is affirmed (1) or not (0)
- name of the role
- length of the name of the role (-1 == calculate it automatically)
- whether to use base-64 encoding (1) or not (0)

Signing through the MS CSP is supported by function `calculateSigInfoSignatureWithCSPEstID()`, use of PKSC#11 driver is supported by the function `calculateSignatureWithEstID()`. These functions add signature to the DigiDoc structure as well.

At the end we save new structure with the function `createSignedDoc()`.

How to sign with mobile phone using MSSP-GW service?

DigiDoc library offers the possibility to sign with a mobile phone using the MSSP Gateway service. Error-checking and less important declarations of variables are omitted for better readability.

```
#include "mssp/DigiDocMsspGw.h"
...
SignedDoc* pSigDoc;
SignatureInfo* pSigInfo;

initDigiDocLib();
initConfigStore(NULL);
...

nPollFreq = ConfigItem_lookup_int("DIGIDOC_MSSP_STATUS_POLL_FREQ",
0);
// send signature req
int err = ddocConfMsspSign(pSigDoc, &g_mssp, szPhoneNo,
                           manifest, city, state, zip, country,
                           p_szOutFile);
  if(!err && nPollFreq) {
    // now check status
    do {
#ifdef WIN32
        time(&tOld);
        do {
                time(&tNew);
        } while(tNew - tOld < nPollFreq);
#else
      sleep(nPollFreq);
#endif
      err = ddocConfMsspGetStatus(&g_mssp, pSigDoc);
    } while(!err && g_mssp.nStatusCode == OUSTANDING_TRANSACTION);
  }

SignedDoc_free(pSigDoc);
cleanupConfigStore(NULL);
finalizeDigiDocLib();
ddocMsspDisconnect(&g_mssp);
```

Function `ddocConfMsspSign()`is used for sending a signature request to MSSP Gateway. Parameters to this function are:

- location of DigiDoc structure
- MSSP Gateway context
- phone number of the signature device
- manifest and signers address

- output file

Function `ddocConfMsspGetStatus()` is used to check the status of signing process and acquire the signature value if the user has already signed or cancelled.

Example programs for using DigiDoc C-library

The following are some illustrative but useful sample programs.

CreateDoc.c

This example program should work both in Linux and Windows environments. The program creates and saves new DigiDoc structure using supplied data file and MIME type. Three parameters are required for input:

- name of a data file
- MIME type of the data file
- output file name

```
#include "DigiDocLib.h"
#include <stdio.h>
#include <string.h>
//-------------------------------------------------
// Use this as a sample how to create a signed doc
// and how to read it.
//-------------------------------------------------
int main(int argc, char** argv)
{
      int rc;
      SignedDoc* pSigDoc;
      DataFile* pDataFile;
      char *file1, *mime1, *outfile;

                char buf[100];
      // print usage...
      if(argc != 4) {
      printf("Usage: CreateDoc <infile> <mime> <outfile> \n");
      return 1;
      }
      file1 = argv[1];
      mime1 = argv[2];
      outfile = argv[3];
      // init DigiDoc library
      initDigiDocLib();
      // create signed doc
      printf("Creating signed doc\n");
      rc = SignedDoc_new(&pSigDoc, SK_XML_1_NAME, SK_XML_1_2_VER);
      // add DataFile1 - embedded PDF file in Base64
      printf("Embedding (Base64) file: %s - %s\n", file1, mime1);
      rc=DataFile_new(&pDatafile, pSigDoc, NULL,file1,
      CONTENT_EMBEDDED_BASE64,
                mime1, 0, NULL, 0, NULL, charset);
      // now calculate file length and digest
      rc = calculateDataFileSizeAndDigest(pSigDoc,
```

```
      pDataFile->szId, file1, DIGEST_SHA1);
      // add some arbitrary attributes
      addDataFileAttribute(pDataFile, "ISBN", "000012345235623465");
      addDataFileAttribute(pDataFile, "Owner", "Veiko");
      // write to file
      printf("Writing to file: %s\n", outfile);
      rc = createSignedDoc(pSigDoc,NULL, outfile);
      // cleanup
      SignedDoc_free(pSigDoc);

      // reading DigiDoc
                rc = readSignedDoc(&pSigDoc, argv[3],0, buf);

      // cleanup of DigiDoc library
      finalizeDigiDocLib();
      return 0;
}
```

ListSignatures.c

This example program demonstrates reading and verifying of signatures and displaying signature data. Following parameters are required for the program as command line options:

- infile = DigiDoc file with one or more signatures
- cafile = certificate of CA who certified signatories
- rootca = root certificate (certifier of CA)
- notcert = OCSP Responder certificate
- capath = directory for additional certificates. If there is no more certififcates required to construct full certification path then this parameter can carry any value.

```
#include "DigiDocLib.h"
#include <stdio.h>
#include <string.h>
//-------------------------------------------------
// Demonstrates how to check all signatures.
// Shows only a CVS list of signatures with status.
//-------------------------------------------------
int main(int argc, char** argv)
{
      int rc, d, i, rc2,rc3;
      SignedDoc* pSigDoc;
      ErrorInfo *pInfo;
      SignatureInfo* pSigInfo;
      NotaryInfo* pNotInfo;
      char *infile, *cafile, *rootca, *notcert, *capath, *role,
      *stime;
      char buf[300],buf2[300];

      // print usage...
      if(argc != 6) {
            printf("Usage: ListSignatures <infile> <CA-cert> <root-
      cert> <notary-cert> <ca-path>\n");
      return 1;
```

```
	}
	infile = argv[1];
	cafile = argv[2];
	rootca = argv[3];
	notcert = argv[4];
	capath = argv[5];

	// init DigiDoc library
	initDigiDocLib();

	rc = readSignedDoc(&pSigDoc, infile,0, buf);
	d = getCountOfSignatures(pSigDoc);
	for(i = 0; i < d; i++) {
		pSigInfo = getSignature(pSigDoc, i);
		rc = verifySignatureInfo(pSigDoc, pSigInfo, cafile,
		infile, 1);
		rc2 = -1;
		pNotInfo = getNotaryWithSigId(pSigDoc, pSigInfo->szId);
		if(pNotInfo)
			rc2 = verifyNotaryInfo(pSigDoc, pNotInfo,
		rootca, cafile, capath, notcert);
		rc3=getSignerCode(pSigInfo, buf2);
		role = (char*)getSignerRole(pSigInfo, 0,0);
		if(!strcmp(charset, CHARSET_UTF_8)) {
		l = strlen(role);
		role = (char*)malloc(l+1);
		ascii2utf8(getSignerRole(pSigInfo, 0,0), role, &l);
		}
		if (!rc3)
		{
			printf("%s/%s/%s/%s/%s/%s/%s\n",
			pSigInfo->szId,
			((!rc) ? "OK" : getErrorString(rc,1)),
			buf2,
			pSigInfo->szTimeStamp,
			((rc2 == -1) ? "NONE" : ((!rc2) ? "OK" :
			getErrorString(rc2,1))), role, pNotInfo-
			>timeProduced);
		}
		if(!strcmp(charset, CHARSET_UTF_8) &&
		role != getSignerRole(pSigInfo, 0,0))
				free(role);
	}
	// cleanup
	SignedDoc_free(pSigDoc);
	// cleanup of DigiDoc library
	finalizeDigiDocLib();
	return 0;
}
```

DigiDoc teek

Andmestruktuurid

SignedDoc

DigiDoc dokumendid on kõik väljendatavad struktuuriga SignedDoc:

```
typedef struct SignedDoc_st {
char* szFormat; // formaadi nimi
char* szFormatVer; // formaadi versioon
int nDataFiles; // andmefailide arv
DataFile** pDataFiles; // andmefailid
int nSignatures; // allkirjade arv
SignatureInfo** pSignatures; // allkirjad
int nNotaries; // kehtivuskinnituste arv
NotaryInfo** pNotaries; // kehtivuskinnitused
} SignedDoc;
```

Üks allkirjastatud fail võib seega sisaldada algandmete faile (või viiteid neile), allkirju ja kehtivuskinnitusi.

DataFile

Algandmete failid on salvestatud järgmise andmestruktuuri abil:

```
typedef unsigned char byte;

typedef struct DataFile_st {
char* szId; // faili unikaalne tunnus
```

```
char* szFileName; // andmefaili nimi

char* szMimeType; // andmetüüp (mime tüüp)

char* szContentType; // DETATCHED, EMBEDDED või EMBEDDED_BASE64

long nSize; // faili suurus (algkujul)

char* szDigestType; // räsi tüüp (sha1)

byte* szDigest; // räsi väärtus

int nDigestLen; // räsi pikkus (20)

int nAttributes; // lisa attribuutide arv

char* szCharset; // datafile initial codepage

char** pAttNames; // attribuutide nimed

char** pAttValues; // attribuutide väärtused

} DataFile;
```

Algandmete fail võib olla sisestatud allkirjastatud dokumenti algkujul (EMBEDDED), Base64 kodeeritud kujul (EMBEDDED_BASE64) või allkirjastatud faili sisestatakse ainult viide välisele failile. Kui algandmefail soovitakse sisestada algkujul, siis tingituna hetkelisest XML-l põhinevast faili formaadist peab tegu olema XML kujul andmetega, mis ei tohi sisaldada siin kasutatud XML tag-e (vaata dokumenti "DigiDoc formaadi kirjeldus") ega XML faili algusrida (<?xml ... ?>). Algkujul XML andmete allkirja kontroll on natuke aeglasem, sest tingituna SAX parserist ei saa neid andmeid terviklikul kujul vaid ainult SAX sündmustena ja seetõttu tuleb räsi arvutamiseks antud faili teistkordselt lugeda. Kui algandmefaili kohta sisestatakse vaid viide, siis peab allkirja kontrollimise ja faili sisselugemise hetkel vastav väline fail olema allkirjastatud dokumendiga samas kataloogis.

Iga algandmefaili kohta kaitstakse allkirjastatud räsi abil vaid faili sisu (algkujul) ja andmetüüp (szMimeType). Muud faili attribuudid on informatiivse sisuga. Faili nimi sisestatakse alati ilma teekonnata. Failile võib lisada suvalise arvu muid atribuute (meta-info jaoks), mis esitatakse nimi/väärtus paaridena. DigiDoc teek haldab nende salvestamist ja lugemist kui ei süüvi sisusse. Nimetatud atribuutide nimed ja väärtused ei tohi sisaldada reavahetusi ja spetsiaalseid XML poolt reserveeritud sümboleid.

Iga allkiri peab kinnitama kõik allkirjastatud dokumendis sisalduvad algandmefailid. Peale esimese allkirja lisamist ei tohi enam

algandmefaile lisada ega muuta. Allkirjad on kirjeldatud järgmiste andmestruktuuridega:

DocInfo

```
Ühe konkreetse algandmefaili allkirjastatud attribuudid


typedef struct DocInfo_st {

char* szDocId; // dokumendi unikaalne tunnus

char* szDigestType; // räsi tüüp (sha1)

byte* szDigest; // algandmete räsi väärtus

int nDigestLen; // algandmete räsi pikkus (20)

byte* szMimeDigest; // andmetüübi räsi väärtus

int nMimeDigestLen; // andmetüübi räsi pikkus (20)

} DocInfo;
```

SignatureProductionPlace

```
Allkirjastamise koht


typedef struct SignatureProductionPlace_st {

char* szCity; // linna nimi

char* szStateOrProvince; // maakonna nimi

char* szPostalCode; // postiindeks

char* szCountryName; // riigi nimi

} SignatureProductionPlace;
```

SignerRole

Allkirjastaja rollid

```
typedef struct SignerRole_st {
int nClaimedRoles; // kinnitamata rollide hulk
char** pClaimedRoles; // kinnitamata rollid
int nCertifiedRoles; // kinnitatud rollide hulk
char** pCertifiedRoles; // kinnitatud rollid
} SignerRole;
```

SignatureInfo

Peamine struktuur mis hoiab kliendi allkirja infot

```
typedef struct SignatureInfo_st {
char* szId; //allkirja unikaalne tunnus
int nDocs; //allkirjastatud algandmefailide hulk
DocInfo** pDocs; // allkirjastatud algandmefailid info
char* szTimeStamp; // allkirjastamise aeg formaadis
// "YYYY-MM-DDTHH:MM:SSZ"
byte* szSigPropDigest;// allkirjastatud allkirja omaduste
// räsi väärtus
```

```
int nSigPropDigestLen;// allkirjastatud allkirja // omaduste räsi
pikkus (20)

byte* szSigPropRealDigest; // allkirjastatud allkirja omaduste räsi
väärtus failist loetud kujul

int nSigPropRealDigestLen; // räsi pikkus (20)

byte* szSigInfoRealDigest; // elemendi <SignedInfo> räsi väärtus
failist loetud kujul

int nSigInfoRealDigestLen; // räsi pikkus (20)

char* szSigType; // allkirja tüüp

// (sha1WithRSAEncryption)

byte* szSigValue; // allkirja väärtus

short nSigLen; // allkirja väärtuse pikkus (128)

void* pX509Cert; // allkirjastaja sertifikaat

long nIssuerSerial; // sertifikaadi number

byte* szCertDigest; // sertifikaadi räsi väärtus

int nCertDigestLen;// sertifikaadi räsi pikkus

SignatureProductionPlace sigProdPlace;

SignerRole signerRole;

} SignatureInfo;
```

NotaryInfo

NotaryInfo ehk allkirjastaja sertifikaadi kehtivuskinnitus. Igal allkirjal võib olla null või enam kehtivuskinnitust.
Kehtivuskinnitused on salvestatud järgmise andmestruktuuri abil:

```
typedef struct NotaryInfo_sk {

char* szId; //kehtivuskinnituse unikaalne tunnus

char* szSigId; //kinnitatud allkirja unikaalne tunnus
```

```
char* szNotType; // kehtivuskinnituse tüüp (OCSP-1.0)

char* timeProduced;// kehtivuskinnituse tekitamise aeg

char* szRespIdType;//kehtivuskinnituse andja tunnuse tüüp

// ("NAME" või "KEY HASH")

char* szRespId; // kehtivuskinnituse andja tunnus

int nRespIdLen; //kehtivuskinnituse andja tunnuse pikkus

char* thisUpdate; // sertifikaadi kinnituse aeg

char* nextUpdate; // sertifikaadi järgmise kinnituse aeg

// (esialgu alati NULL)

unsigned long certNr; // sertifikaadi number

char* szDigestType; // räsi tüüp (sha1)

byte* szIssuerNameHash;//sertifikaadi väljaandja nime räsi

int nIssuerNameHashLen;// sertifikaadi väljaandja nime

// räsi pikkus

byte* szIssuerKeyHash;//sertifikaadi väljaandja võtme räsi

int nIssuerKeyHashLen;//sertifikaadi väljaandja võtme

//räsi pikkus

byte* szUserSign; // kinnitatud allkirja väärtuse räsi // (OCSP
Nonce)

int nUserSignLen; // kinnitatud allkirja allkirja

// väärtuse räsi pikkus

// notaries personal signature

char* szSigType; // kehtivuskinnituse allkirja tüüp

// (sha1WithRSAEncryption)

byte* szSigValue; // kehtivuskinnituse allkirja väärtus

short nSigLen;//kehtivuskinnituse allkirja väärtuse pikkus

void* pX509Cert;// kehtivuskinnituse andja sertifikaat

           long nIssuerSerial;//kehtivuskinnituse andja sertifikaadi
```

```
                                //number
byte* szCertDigest; // kehtivuskinnituse andja
                                //sertifikaadi räsi
          int nCertDigestLen; // kehtivuskinnituse andja
                                //sertifikaadi räsi pikkus
} NotaryInfo;
```

Timestamp

Seda struktuuri kasutatakse erinevate aegade arvutamisel DigiDoc'is.

```
typedef struct Timestamp_st {
int year; //aasta
int mon; //kuu
int day; //päev
int hour; //tunnid
int min; //minutid
int sec; //sekundid
int tz; //ajavööde GMT'st.
} Timestamp;
```

PolicyIdentifier

Seda struktuuri kasutatakse sertifikaadi kasutamise ja allkirjastamise reeglite lugemiseks allkirjastaja sertifikaadist.

```
typedef struct PolicyIdentifier_st {
char* szOID; // stringikujuline reegli OID
char* szCPS; // sertfikaadi reegli URL
char* szUserNotice; // märkus / kommentaar
} PolicyIdentifier;
```

CertSearch

Seda struktuuri kastutatakse sertifikaatide otsimiseks ja lugemiseks.

```
typedef struct CertSearch_st {
int searchType;//otsingu tüüp v.t "Kasutusel olevad konstantid"
char* x509FileName; //Kui PEM fail siis selle nimi
char* keyFileName; // Privaatvõtme faili nimi
char* pkcs12FileName; //PKCS #12 konteineri nimi
char * pswd; // privaatvõtme/ PKCS #12 parool
CertSearchStore* certSearchStore; //MS sertifikaatde hoidla otsing
} CertSearch;
```

CertSearchStore

MS sertifikaatide hoidla otsingu kriteeriumite struktuur.

```
typedef struct CertSearchStore_st {
int searchType; // mille järgi saab otsida
char* storeName; // default is "My"
long certSerial; // serdi seerinumber
int numberOfSubDNCriterias; //subjekti nime komponentide arv
char** subDNCriterias; // subjekti nime komponendid
```

```
int numberOfIssDNCriterias; // väljaandja nime komponentide arv
char** issDNCriterias; // väljaandja nime komponendid
void* publicKeyInfo; // avaliku võtme info} CertSearchStore;
```

ErrorInfo

Veainfot esitatakse mälus ErrorInfo struktuurina:

```
typedef struct ErrorInfo_st {
int code; //veakood
char *fileName; //lähtekoodi fail, kus viga avastati
int line; //lähtekoodi rea number, kus viga avastati
char *assertion;//vääraks osutunud kontrollavaldis
} ErrorInfo;
```

CSProvider

Hoiab teegi jaoks vajalikke CSP parameetreid.

```
typedef struct CSProvider_st {
char* CSPName;
int rsa_full; // kui FALSE, siis kasutatakse RSA_SIG
int at_sig; // kui FALSE, siis kasutatakse AT_KEYEXCHANGE
} CSProvider;
```

CertItem

Üldine struktuur X509 viitade nimistu hoidmiseks:

```
typedef struct CertItem_st {
```

```
X509* pCert;

struct CertItem_st* nextItem;

} CertItem;
```

DEncEncryptionProperty

Seda struktuuri kasutatakse krüpteeritud dokumendi mingite algsete omaduste salvestamiseks mis krüpteeritud kujul ei salvestata. Näiteks faili algne nimi, suurus ja mime tüüp.

```
typedef struct DEncEncryptionProperty_st {

char* szId; // Id atribuudi väärtus (optional)

char* szTarget; // Target atribuudi väärtus (optional)

char* szName; // "name" atribuudi väärtus (vajalik)

char* szContent; // elemendi sisu

} DencEncryptionProperty;
```

DEncEncryptionProperties

Seda struktuuri kasutatakse krüpteeritud dokumendi omaduste loeteluna.

```
typedef struct DEncEncryptionProperties_st {

char* szId; // ID atribuudi väärtus (optional)

DEncEncryptionProperty** arrEncryptionProperties; // loetelu

int nEncryptionProperties; // elemendtide arv

} DEncEncryptionProperties;
```

DEncEncryptedKey

Seda struktuuri kasutatakse krüpteeritud dokumendi vastuvõtja andmete salvestamiseks

```
typedef struct DEncEncrytedKey_st {

char* szId; // Id atribuudi väärtus (optional)

char* szRecipient; // Recipient atribuudi väärtus (optional)

char* szEncryptionMethod; // EncryptionMethod elemendi väärtus
(vajalik)

char* szKeyName; // KeyName elemendi väärtus (optional)

char* szCarriedKeyName; // CarriedKeyName elemendi väärtus (optional)

X509* pCert; // vastuvõtja sertifikaat (vajalik)

DigiDocMemBuf mbufTransportKey; // krüpteeritud AES transpordivõti

} DEncEncryptedKey;
```

DEncEncryptedData

Seda struktuuri kasutatakse krüpteeritud dokumendi andmete hoidmiseks.

```
typedef struct DEncEncrytedData_st {

char* szId; // Id atribuudi väärtus (optional)

char* szType; // Type atribuudi väärtus (optional)

char* szMimeType; // MimeType atribuudi väärtus (optional)

char* szEncryptionMethod; // EncryptionMethod elemendi väärtus
(nõutud)

char* szXmlNs; // xmlns atribuudi väärtus (optional)

DEncEncryptedKey ** arrEncryptedKeys; // <EncryptedKey> loetelu

int nEncryptedKeys; // loetelu pikkus
```

```
DigiDocMemBuf mbufEncryptedData; // krüpteeritud andmed

DEncEncryptionProperties encProperties; // algfaili omadused

// private transient fields

DigiDocMemBuf mbufTransportKey; // krüpteerimata transpordivõti

// flags

int nDataStatus;

int nKeyStatus;

} DEncEncryptedData;
```

DEncRecvInfo

Seda struktuuri kasutatakse krüpteeritud dokumendi vastuvõtja andmete haldamiseks ja salvestamiseks konfiguratsioonifailides.

```
typedef struct DEncRecvInfo_st {

char* szId; // ID atribuudi väärtus (vajalik)

char* szRecipient; // Recipient atribuudi väärtus (optional)

char* szKeyName; // KeyName elemendi väärtus (optional)

char* szCarriedKeyName; // CarriedKeyName elemendi väärtus (optional)

X509* pCert; // vastuvõtja sertifikaat (vajalik)

} DEncRecvInfo;
```

DEncRecvInfoList

Seda struktuuri kasutatakse RecvInfo struktuuride loeteluna.

```
typedef struct DEncRecvInfoList_st {

int nItems; // objektide hulk
```

```
DEncRecvInfo** pItems; // loetelu

} DEncRecvInfoList;
```

Üldised - ja administreerimisfunktsioonid

Sellesse kategooriasse kuuluvad DigiDoc teegi initsialiseerimis-, sulgemis- ja versiooniinfo funktsioonid.

getLibName()

```
Tagastab teegi nime

const char* getLibName();
```

getLibVersion()

```
Tagastab teegi versiooni

const char* getLibVersion();
```

getSupportedFormats()

```
Tagastab NULL-ga lõpetatud massiivi toetatud formaatidest

const char** getSupportedFormats();
```

getSupportedFormatsAndVersions()

```
Tagastab NULL-ga lõppeva jada toetatud formaadi ja versiooni
kombinatsioonidest.

FormatAndVer* getSupportedFormatsAndVersions();
```

initDigiDocLib()

```
Initsialiseerib DigiDoc teegi ja selle poolt kasutatavad teegid
(OpenSSL), tuleb alati programmi algul välja kutsuda

void initDigiDocLib();
```

finalizeDigiDocLib()

```
Suleb (cleanup) DigiDoc teegi ja kasutusel olnud teegid (OpenSSL),
tuleb alati programmi lõpus välja kutsuda

void finalizeDigiDocLib();
```

trim()

Eemaldab algusest ja lõpust tühikud, ning

reavahetused - ' ','\n','\r'

```
char* trim(char* src);
```

ascii2utf8()

Funktsioon, mis konverteerib esimese parameetris antud ASCII (ISO Latin1) stringi teises parameetris allokeeritud mällu UTF8 vormingusse.

- ascii - sisend andmed ISO Latin1 vormingus
- utf8out - output puhver UTF8 teksti jaoks
- outlen - output puhvri pikkus

```
char* ascii2utf8(const char* ascii, char* utf8out, int* outlen);
```

utf82ascii()

Eelmise pöördfunktsioon, konverteerib esimeses parameetris antud UTF8 stringi teises parameetris allokeeritud mällu ASCII (ISO Latin1) vormingusse.

- utf8in - sisend andmed UTF8 vormingus
- asciiout - väljund puhver ISO Latin1 vormingus stringi jaoks
- outlen - väljund puhveri pikkus

```
char* utf82ascii(const char* utf8in, char* asciiout, int* outlen);
```

unicode2ascii()

Konverteerib stringi unicode'ist asciisse

- uni - sisend string unicode'is
- dest - ascii vormingus väljundile allokeeritud puhver

void unicode2ascii(const char* uni, char* dest);

convertStringToTimestamp()

Teisendab stringi kujul antud Timestamp'i vastavaks struktuuriks.

- pSigDoc - viide kasutusel olevale DigiDoc'ile

- szTimestamp - Timestamp stringi kujul
- pTimestamp - Timestamp struktuur kuhu fuktsioon kirjutab oma väljundi

```
void convertStringToTimestamp(const SignedDoc* pSigDoc,const char*
szTimestamp, Timestamp* pTimestamp);
```

convertTimestampToString()

Teisendab antud Timestamp struktuuri stringiks

- pSigDoc - viide kasutusel olevale DigiDoc'ile

- pTimestamp - Timestamp struktuur milest luuakse string
- szTimestamp - allokeeritud puhver Timestamp stringi kirjutamiseks

```
void convertTimestampToString(const SignedDoc* pSigDoc, const
Timestamp* pTimestamp, char* szTimestamp);
```

Timestamp_new()

Konstrueerib antud parameetritest uue Timestamp struktuuri.

- ppTimestamp – viit viidale, mis saab viitama loodud struktuurile.
- year - aasta
- month - kuu
- day - kuupäev
- hour - tunnid
- minute - minutud
- second - sekundid
- timezone – ajavöönd

Funktsioon tagastab ERR_OK või veakoodi.

```
int Timestamp_new(Timestamp **ppTimestamp, int year,int month,int
day,int hour,int minute,int second,int timezone)
```

Timestamp_free()

Kustutab antud Timestamp struktuuri ja vabastab mälu.

```
void Timestamp_free(Timestamp* pTimestamp);
```

convertStringToTimeT ()

Konverteerib ajatempli stringi time_t väärtuseks.

```
time_t convertStringToTimeT(const SignedDoc* pSigDoc, const char*
szTimestamp);
```

Üldised krüptograafiafunktsioonid

Selle kategooria funktsioone kasutatakse enamasti teegi siseselt, kuid võivad ka mujal kasulikuks osutuda.

bin2hex()

Konverteerib binaarsed andmed vastavaks hex-stringiks

- pData – algandmed
- nDataLen - algandmete pikkus
- pBuf - buffer hex-stringi jaoks. Vaja on algandmetest 2 korda pikemat puhvrit
- nBufLen - puffri suuruse aadress. Tuleb enne funktsiooni kasutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia kasutatud pikkuse.

Funktsiooni tulemuseks on veakood või 0 (ERR_OK)

```
int bin2hex(const byte* pData, int nDataLen,

char* pBuf, int* nBufLen);
```

calculateFileDigest()

Arvutab faili SHA1 räsikoodi

- szFileName – faili nimi
- nDigestType – räsi tüüp. Peab olema – DIGEST_SHA1
- pDigestBuf – puffer räsi väärtuse jaoks
- nDigestLen - puffri suuruse aadress. Tuleb enne funktsiooni asutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia asutatud pikkuse.
- lFileLen – siia salvestatkse loetud faili pikkus baitides

```
int calculateFileDigest(const char* szFileName, int nDigestType,

byte* pDigestBuf, int* nDigestLen, long* lFileLen);
```

calculateFileSignature()

Arvutab faili RSA+SHA1 allkirja

- szFileName – faili nimi
- nDigestType – räsi tüüp. Peab olema – DIGEST_SHA1
- pSigBuf – puffer allkirja väärtuse jaoks
- nSigLen - puffri suuruse aadress. Tuleb enne funktsiooni kasutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia asutatud pikkuse.
- keyfile – allkirjastaja salajase võtme faili nimi (PEM formaadis)
- passwd - allkirjastaja salajase võtme salakood (ei toimi hetkel!)

```
int calculateFileSignature(const char* szFileName, int nDigestType,

byte* pSigBuf, int* nSigLen, const char *keyfile,

const char* passwd);
```

verifyFileSignature()

Kontrollib faili RSA+SHA1 allkirja

- szFileName – faili nimi
- nDigestType – räsi tüüp. Peab olema – DIGEST_SHA1
- pSigBuf – allkirja väärtuse
- nSigLen - allkirja väärtuse pikkus
- certfile – allkirjastaja sertifikaadi faili nimi (PEM formaadis)

```
int verifyFileSignature(const char* szFileName, int nDigestType,
```

```
byte* pSigBuf, int nSigLen, const char *certfile);
```

signData()

- Arvutab mingite andmete RSA+SHA1 allkirja
- data – allkirjastatavad andmed
- dlen - allkirjastatavate andmete pikkus
- nDigestType – räsi tüüp. Peab olema – DIGEST_SHA1
- pSigBuf – puffer allkirja väärtuse jaoks
- nSigLen - puffri suuruse aadress. Tuleb enne funktsiooni kasutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia asutatud pikkuse.
- keyfile – allkirjastaja salajase võtme faili nimi (PEM formaadis)
- passwd - allkirjastaja salajase võtme salakood (ei toimi hetkel!)

```
int signData(const byte* data, int dlen, byte* pSigBuf, int* nSigLen,

int nDigestType, const char *keyfile, const char* passwd);
```

calculateDigest()

Arvutab mingite andmete SHA1 räsikoodi

- data – allkirjastatavad andmed
- dlen - allkirjastatavate andmete pikkus
- nDigestType – räsi tüüp. Peab olema – DIGEST_SHA1
- pDigestBuf – puffer räsi väärtuse jaoks
- nDigestLen - puffri suuruse aadress. Tuleb enne funktsiooni asutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia asutatud pikkuse.

```
int calculateDigest(const byte* data, int nDataLen, int nDigestType,

byte* pDigestBuf, int* nDigestLen);
```

encode()

Kodeerib sisendandmed Base64 kujul

- raw – algandmed
- rawlen – algandmete pikkus
- buf – puffer Base64 andmete jaoks

- buflen - puffri suuruse aadress. Tuleb enne funktsiooni kasutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia asutatud pikkuse.

```
void encode(const byte* raw, int rawlen, byte* buf, int* buflen);
```

decode()

Dekodeerib Base64 kujul sisendandmed

- raw – Base64 kujul algandmed
- rawlen – Base64 kujul alganmete pikkus
- buf – puffer dekodeeritud andmete jaoks
- buflen - puffri suuruse aadress. Tuleb enne funktsiooni kasutamist initsialiseerida puffri pikkusega. Funktsioon salvestab siia asutatud pikkuse.

```
void decode(const byte* raw, int rawlen, byte* buf, int* buflen);
```

Üldised XMLi genereerivad funktsioonid

Need funktsiooni tekitavad XML-i vorme mida teegi allkirjastamise ja räsifunktsiooni funktsioonid kasutavad sisendandmeteks.

createTimestamp()

Konverteerib arvuti jooksva ajahetke kujul `DD.MM.YYYY HH:MM:SS+HH:MM` puhvrisse buf, mis peab olema eelnevalt allokeeritud. Tagastab kirjutatud ajatempli pikkuse.

- buf - puhhver ajatempli salvestamiseks.

```
int createTimestamp(char* buf);
```

createXMLSignedInfo()

Loob `<SignedInfo>` XML bloki antud sisendist ja tagastab selle stringina.

- pSigInfo - SignedDoc objekt
- `pSigInfo` - allkirja info objekt

```
char* createXMLSignedInfo(const SignedDoc* pSigDoc, const
SignatureInfo* pSigInfo);
```

createMimeType ()

Loob MimeType="" atribuudi allkirja arvutamiseks. Kasutatid vanemas 1.0 versioonis.

```
int createMimeType(char* buf, const char* mime, const char* sigId,
const char* docId);
```

Allkirjastatud dokumendi halduse funktsioonid

Selle kategooria funktsioonid haldavad ülalkirjeldatud andmestruktuuride elemente, allokeerivad mälu uute jaoks jne.

getSimpleFileName()

Hangib faili nime (ilma teekonnata), ehk teisiti öeldes eraldab failinime teekonnast. Tagastab pointeri mis viitab sellele aadressile parameetriks antud stringi sees kust hakkab failinime osa.

- szFileName – faili nimi koos teekonnaga

```
const char* getSimpleFileName(const char* szFileName);
```

SignedDoc_new()

Allokeerib uue allkirjastatud dokumendi struktuuri

- pSignedDoc – uus loodav struktuur.
- format – allkirjastatud dokumendi formaat. Peab olema – SK_XML_1_NAME
- version – formaadi versioon. Peab olema - SK_XML_1_2_VER

```
int SignedDoc_new(SignedDoc **pSignedDoc, const char* format, const
char* version)
```

SignedDoc_free()

Vabastab antud allkirjastatud dokumendi ja kõigi tema osade jaoks allokeeritud mälu.

- pSigDoc - allkirjastatud dokumendi struktuuri aadress

```
void SignedDoc_free(SignedDoc* pSigDoc);
```

getCountOfDataFiles()

Tagastab allkirjastatud dokumendis registreeritud algandmefailide arvu.

- pSigDoc - allkirjastatud dokumendi struktuuri aadress

```
int countDataFiles(const SignedDoc* pSigDoc);
```

getDataFile()

Tagastab soovitud algandmefaili info

- pSigDoc - allkirjastatud dokumendi struktuuri aadress
- nIdx – algandmefaili indeks (algab 0-st)

```
DataFile* getDataFile(const SignedDoc* pSigDoc, int nIdx);
```

getDataFileWithId()

Tagastab soovitud tunnusega algandmefaili info

- pSigDoc - allkirjastatud dokumendi struktuuri aadress
- id – algandmefaili unikaalne tunnus

```
DataFile* getDataFileWithId(const SignedDoc* pSigDoc, const char* id);
```

ddocGetDataFileCachedData()

Tagastab soovitud algandmefaili sisu mälupuhvrist kui võimalik, s.o. Kui antud algandmefaili andmeid hoitakse mälus.

- pSigDoc - allkirjastatud dokumendi struktuuri aadress
- szDocId – algandmefaili unikaalne tunnus
- ppBuf – aadress allokeeritava mälu aadressi jaoks. Kasutaja peab mälu vabastama.
- pLen – allokeeritud puhvri pikkuse aadress.

```
int ddocGetDataFileCachedData(SignedDoc* pSigDoc, const char*
szDocId, void** ppBuf, int* pLen);
```

ddocAppendDataFileData()

Seda funktsiooni kasutatakse sisemiselt algandmete lugemisel ja lisamisel mälupuhvrisse.

- pDf – algandmefaili objekt
- maxLen – suurim algandmefaili suurus mille puhul andmeid veel hoitakse mälus.
- data – lisatavad andmed
- len – lisatavata andmete pikkus baitides.

```
void ddocAppendDataFileData(DataFile* pDf, int maxLen, void* data,
int len);
```

ddocGetLastDataFile()

Tagastab viimase angandmefaili info.

- pSigDoc - signed doc pointer

```
DataFile* ddocGetLastDataFile(const SignedDoc* pSigDoc);
```

ddocGetDataFileFilename()

Tagastab soovitud angandmefaili Filename atribuudi väärtuse. Parandab formaatides 1.0, 1.1 ja 1.2 kasutatud vale UTF-8 kodeeringu.

- pSigDoc - signed doc objekt
- szDocId – andmefaili tunnus
- ppBuf – aadress allokeeritud mälu pointeri jaoks.
- pLen – aadress puhvri pikkuse salvestamiseks.

```
int ddocGetDataFileFilename(SignedDoc* pSigDoc, const char* szDocId,
void** ppBuf, int* pLen);
```

DataFile_new()

Allokeerib mälu uue algandmefaili info jaoks. Iga parameetri jaoks, mille väärtust hetkel ei tea tuleb kasutada NULL-i ja vastava parameetri pikkus, kui see on nõutud, peab siis olema 0.

- newDataFile – loodav struktuur.
- pSigDoc - allkirjastatud dokumendi struktuuri aadress
- id – algandmefaili unikaalne tunnus. Kasuta NULL-i vaikimis omistatud tunnuse kasutamiseks (soovitatud variant)
- filename – algandmefaili nimi
- contentType – faili tüüp (mime tüüp)
- size – faili suurus baitides
- digest – faili räsikood
- digLen - faili räsikoodi pikkus
- digType - faili räsikoodi tüüp. Kasuta - DIGEST_SHA1_NAME
- szCharset - sisendandmete kooditabel. Kasuta CHARSET_ISO_8859_1 või CHARSET_UTF_8
- szFileNameCharset - faili nime kooditabel. Kasuta CHARSET_ISO_8859_1 või CHARSET_UTF_8

Funktisoon tagastab ERR_OK või veakoodi.

```
int DataFile_new(DataFile **newDataFile, SignedDoc* pSigDoc,

const char* id, const char* filename,

const char* contentType,

const char* mime, long size,

const byte* digest, int digLen,

const char* digType, const char* szCharset,

const char* szFileNameCharset)
```

DataFile_free()

Vabastab antud algandmefaili info jaoks allokeeritud mälu

- pDataFile - algandmefaili info aadress

```
void DataFile_free(DataFile* pDataFile);
```

getCountOfDataFileAttributes()

Tagastab algandmefaili lisa-atribuutide arvu

- pDataFile - algandmefaili info aadress

```
int countDataFileAttributes(const DataFile* pDataFile);
```

addDataFileAttribute()

Lisab algandmefailile mingi uue attribuudi

- pDataFile - algandmefaili info aadress
- name – attribuudi nimi
- value – attribuudi väärtus

```
void addDataFileAttribute(DataFile* pDataFile, const char* name,
const char* value);
```

getDataFileAttribute()

Tagastab algandmefailile soovitud attribuudi

- pDataFile - algandmefaili info aadress
- idx – attribuudi indeks
- name – puffer attribuudi nime aadressi jaoks
- value – puffer attribuudi väärtuse aadressi jaoks

```
void getDataFileAttribute(DataFile* pDataFile, int idx,

char** name, char** value);
```

calculateDataFileSizeAndDigest()

Arvutab soovitud algandmefaili suuruse ja räsikoodi ning salvestab need andmed allkirjastatud dokumendi struktuuri.

- pSigDoc – allkirjastatud dokumendi info
- id – algandmefaili tunnus
- filename – algandmefaili nimi
- digType – räsikoodi tüüp. Peab olema - DIGEST_SHA1

```
int calculateDataFileSizeAndDigest(SignedDoc* pSigDoc,

const char* id, const char* filename, int digType);
```

getCountOfSignatures()

Tagastab allkirjade arvu

- pSigDoc – allkirjastatud dokumendi info

```
int countSignatures(const SignedDoc* pSigDoc);
```

getSignature()

Tagastab soovitud allkirja info

- pSigDoc – allkirjastatud dokumendi info
- nIdx – allkirja indeks

```
SignatureInfo* getSignature(const SignedDoc* pSigDoc, int nIdx);
```

getSignatureWithId()

Tagastab soovitud tunnusega allkirja info

- pSigDoc – allkirjastatud dokumendi info
- id – allkirja tunnus

```
SignatureInfo* getSignatureWithId(const SignedDoc* pSigDoc, const
char* id);
```

ddocGetLastSignature()

Tagastab viimase allkirja info

- pSigDoc – allkirjastatud dokumendi info

```
SignatureInfo* ddocGetLastSignature(const SignedDoc* pSigDoc);
```

SignatureInfo_new()

Allokeerib mälu uue allkirja struktuuri jaoks

- newSignatureInfo – viit loodava struktuuri viidale.
- pSigDoc – allkirjastatud dokumendi info
- id – allkirja tunnus

```
int SignatureInfo_new(SignatureInfo **newSignatureInfo, SignedDoc*
pSigDoc, const char* id)
```

setSignatureProductionPlace()

Lisab/muudab allkirjastamise koha info. Kasutage NULL-i tundmatute väärtuste jaoks.

- pSigInfo – allkirja info aadress
- city – linna nimi
- state – maakonna nimi
- zip – postiindeks
- country – riigi nimi

```
void setSignatureProductionPlace(SignatureInfo* pSigInfo,

const char* city, const char* state,

const char* zip, const char*);
```

addSignerRole()

Lisab uue allkirjastaja rolli

- pSigInfo – allkirja info aadress
- nCertified – flag: 0=kinnitamata roll, 1=kinnitatud roll (rolli sert)
- role – kinnitamata rolli nimi või kinnitatud rolli serdi aadress
- rLen – stringi puhul -1, serdi puhul serdi andmete pikkus
- encode – flag: 0=salvesta algkujul (string), 1=kodeeri base64 formaadis

```
void addSignerRole(SignatureInfo* pSigInfo, int nCertified,

const char* role, int rLen, int encode);
```

getCountOfSignerRoles()

Tagastab allkirjastaja rollide hulga

- pSigInfo – allkirja info aadress
- nCertified – flag: 0=kinnitamata roll, 1=kinnitatud roll (rolli sert)

```
int countSignerRoles(SignatureInfo* pSigInfo, int nCertified);
```

getSignerRole()

Tagastab soovitud allkirjastaja rolli info

- pSigInfo – allkirja info aadress
- nCertified – flag: 0=kinnitamata roll, 1=kinnitatud roll (rolli sert)
- nIdx – allkirjastaja rolli indeks (kinnitatud ja kinnitamata rollidel on erladi loendurid)

```
const char* getSignerRole(SignatureInfo* pSigInfo, int nCertified,
int nIdx);
```

SignatureInfo_delete()

Kustutab id'ga viidatud SignatureInfo stuktuuri pSigDoc struktuurist ja vabastab tema mälu.

- pSigDoc - signed doc objekt
- id - eemaldatava allkirja id

```
int SignatureInfo_delete(SignedDoc* pSigDoc, const char* id);
```

SignatureInfo_free()

Vabastab allkirja ja tema alamelementide jaoks allokeeritud mälu

- pSigInfo – allkirja info aadress

```
void SignatureInfo_free(SignatureInfo* pSigInfo);
```

addDocInfo()

Lisab allkirja infole uue algandmefaili allkirjastatud atribuutide info.

- pSigInfo – allkirja info aadress
- docId – algandmefaili info tunnus
- digType – räsi tüübi nimi. Peab olema – DIGEST_SHA1_NAME
- digest – faili räsikood
- digLen - faili räsikoodi pikkus
- mimeDig – faili andmetüübi räsikood
- mimeDigLen - faili andmetüübi räsikoodi pikkus

```
DocInfo* addDocInfo(SignatureInfo* pSigInfo, const char* docId,

const char* digType, const byte* digest,

int digLen, const byte* mimeDig, int mimeDigLen);
```

DocInfo_free()

Vabastab algandmefaili allkirjastatud atribuutide info jaoks allokeeritud mälu

- pDocInfo - algandmefaili allkirjastatud atribuutide info aadress

```
void DocInfo_free(DocInfo* pDocInfo);
```

getCountOfDocInfos()

Tagastab algandmefaili allkirjastatud atribuutide info struktuuride hulga

- pSigInfo – allkirja info aadress

```
int countDocInfos(const SignatureInfo* pSigInfo);
```

getDocInfo()

Tagastab algandmefaili allkirjastatud atribuutide info

- pSigInfo – allkirja info aadress
- idx – algandmefaili allkirjastatud atribuutide info indeks

```
DocInfo* getDocInfo(const SignatureInfo* pSigInfo, int idx);
```

getDocInfoWithId()

Tagastab soovitud algandmefaili allkirjastatud atribuutide info

- pSigInfo – allkirja info aadress
- id – algandmefaili tunnus

```
DocInfo* getDocInfoWithId(const SignatureInfo* pSigInfo, const char*
id);
```

ddocGetLastDocInfo()

Tagastab viimase algandmefaili allkirjastatud atribuutide info

- pSigInfo – allkirja info aadress

```
DocInfo* ddocGetLastDocInfo(const SignatureInfo* pSigInfo);
```

setDocInfoDigest()

Seab algandmefaili infos faili räsikoodi väärtuse ja tüübi

- pDocInfo – algandmefaili allkirjastatud info aadress
- digest - algandmefaili räsi väärtus
- digLen - algandmefaili räsi väärtuse pikkus
- digType - algandmefaili räsi tüüp. Kasuta - DIGEST_SHA1_NAME

```
void setDocInfoDigest(DocInfo* pDocInfo, const byte* digest,

int digLen, const char* digType);
```

setDocInfoMimeDigest()

Seab algandmefaili infos andmetüübi räsikoodi väärtuse

- pDocInfo – algandmefaili allkirjastatud info aadress
- mimeDig - algandmefaili andmetüübi räsi väärtus
- mimeDigLen - algandmefaili andmetüübi räsi väärtuse pikkus

```
void setDocInfoMimeDigest(DocInfo* pDocInfo,

const byte* mimeDig, int mimeDigLen);
```

addAllDocInfos()

Arvutab kõigi registreeritud algandmefailide allkirjastatavate atribuutide räsid ja lisab need antud allkirjale

- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info

```
void addAllDocInfos(SignedDoc* pSigDoc, SignatureInfo* pSigInfo);
```

calculateSigInfoSignature()

Arvutab SHA1+RSA allkirja väärtuse ja salvestab ta antud allkirja infos. See ei ole EstID-ga ühilduv allkiri!!!

- pSigDoc – allkirjastatud dokumendi info

- pSigInfo – allkirja info
- nSigType – allkirja tüüp. Peab olema – SIGNATURE_RSA
- keyfile – allkirjastaja salajase võtme faili nimi (PEM)
- passwd - allkirjastaja salajase võtme salasõna
- certfile – allkirjastaja sertifikaadi faili nimi (PEM)

```
int calculateSigInfoSignature(SignedDoc* pSigDoc, SignatureInfo*
pSigInfo, int nSigType, const char* keyfile, const char* passwd,
const char* certfile);
```

getCountOfNotaryInfos()

Tagastab allkirjastatud dokumendi kehtivuskinnituste arvu.

- pSigDoc – allkirjastatud dokumendi inf

```
int countNotaryInfos(const SignedDoc* pSigDoc);
```

getNotaryInfo()

Tagastab soovitud kehtivuskinnituse info

- pSigDoc – allkirjastatud dokumendi info
- nIdx – kehtivuskinnituse indeks

```
NotaryInfo* getNotaryInfo(const SignedDoc* pSigDoc, int nIdx);
```

getNotaryWithId()

Tagastab soovitud tunnusega kehtivuskinnituse info

- pSigDoc – allkirjastatud dokumendi info
- id – kehtivuskinnituse tunnus

```
NotaryInfo* getNotaryWithId(const SignedDoc* pSigDoc, const char*
id);
```

getNotaryWithSigId()

Tagastab soovitud allkirja kehtivuskinnituse info

- pSigDoc – allkirjastatud dokumendi info
- sigId – allkirja tunnus

```
NotaryInfo* getNotaryWithSigId(const SignedDoc* pSigDoc, const char*
sigId);
```

ddocGetLastNotaryInfo()

Tagastab viimase allkirja kehtivuskinnituse info

- pSigDoc – allkirjastatud dokumendi info

```
NotaryInfo* ddocGetLastNotaryInfo(const SignedDoc* pSigDoc);
```

NotaryInfo_new()

Allokeerib mälu uue kehtivuskinnituse jaoks

- newNotaryInfo – viit loodava struktuuri viidale
- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info

```
int NotaryInfo_new(NotaryInfo **newNotaryInfo, SignedDoc* pSigDoc,
const SignatureInfo* pSigInfo)
```

NotaryInfo_new_file()

Allokeerib mälu uue kehtivuskinnituse jaoks ja initsialiseerib ta andmetega OCSP vastuse failist

- newNotaryInfo – viit loodava struktuuri viidale
- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info
- ocspRespFile – OCSP vastuse faili nimi
- notaryCertFile – OCSP responderi sertifikaadi faili nimi (PEM)

Funktsioon tagastab ERR_OK või veakoodi.

```
int NotaryInfo_new_file(NotaryInfo **newNotaryInfo,

SignedDoc *pSigDoc,

const SignatureInfo *pSigInfo,

const char *ocspRespFile,
```

```
const char *notaryCertFile)
```

NotaryInfo_free()

Vabastab antud kehtivuskinnituse jaoks allokeeritud mälu

- pNotary – kehtivuskinnituse info

```
void NotaryInfo_free(NotaryInfo* pNotary);
```

NotaryInfo_delete()

Eemaldab antud id'ga viidatud NotaryInfo antud SignedDoc struktuurist ning vabastab tema mälu.

- pSigDoc - SignedDoc struktuur
- id - eemaldatava notari ifno id

```
int NotaryInfo_delete(SignedDoc* pSigDoc, const char* id);
```

createXMLSignedProperties()

Koostab XML-i <SignedInfo> bloki. Vajalik juhul kui soovite ise allkirja koostada. Siis oleksid need siin andmed, mida allkirjastada

- pSigInfo – allkirja info

```
char* createXMLSignedProperties(const SignatureInfo* pSigInfo);
```

calculateSignedPropertiesDigest()

Arvutab SHA1 räsi XML-i <SignedProperties> blokist. Tagastab veakoodi või 0 (ERR_OK).

- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info

```
int calculateSignedPropertiesDigest(SignedDoc* pSigDoc,
SignatureInfo* pSigInfo);
```

calculateSignedInfoDigest()

Arvutab SHA1 räsi XML'i <SignedInfo> blokist. Tagastab veakoodi või 0 (ERR_OK).

- pSigDoc – allkirjastatud dokumendi info

- pSigInfo – allkirja info
- digBuf - puhver räsi kirjutamiseks
- digLen - algselt puhvri digBuf pikkus, funktsioon uuendab selle räsi tegelikuks pikkuseks

```
int calculateSignedInfoDigest(SignedDoc* pSigDoc, SignatureInfo* pSigInfo, byte* digBuf, int* digLen)
```

setSignatureCertFile()

Loeb failist sertifikaadi arvutab selle räsi ning lisab, sertifikaadi, tema seerianumbri ja räsi antud allkirja info struktuuri.

- pSigInfo - Allkirja info objekt

- certFile - sertifikaadi fail PEM vormingus

```
int setSignatureCertFile(SignatureInfo* pSigInfo, const char* certFile);
```

setSignatureCert()

Arvutab sertifikaadi räsi ning lisab sertifikaadi, tema seerianumbri ja räsi antud allkirja info struktuuri.

- pSigInfo - Allkirja info objekt

- cert - sertifikaat

```
int setSignatureCert(SignatureInfo* pSigInfo, X509* cert);
```

setSignatureValueFromFile()

Loeb antud failist allkirja base64 kujul ja lisab selle antud allkirja info struktuuri.

- pSigInfo - allkirja info struktuur
- szSigFile - faili nimi

```
int setSignatureValueFromFile(SignatureInfo* pSigInfo, char* szSigFile);
```

setSignatureValue()

Määrab allkirja väärtuse.

- pSigInfo - allkirja info struktuur
- szSignature - allkirja väärtus
- sigLen - allkirja pikkus

```
int setSignatureValue(SignatureInfo* pSigInfo, byte* szSignature, int sigLen);
```

Allkirjastatud dokumendi kirjutamine

Selle kategooria funktsioonide abil saab allkirjastatud dokumente digidoc failidesse kirjutada.

createSignedDoc()

Salvestab allkirjastatud dokumendi soovitud faili

- pSigDoc – allkirjastatud dokumendi info
- szOutputFile – faili nimi kuhu salvestada

```
int createSignedDoc(SignedDoc* pSigDoc, const char* szOutputFile);
```

Allkirjastatud dokumendi lugemine SAX parseri abil.

Selle kategooria funktsioonide abil saab allkirjastatud dokumente failist lugeda ja eraldi faili salvestada. Antud moodul kasutab libxml2 teegi SAX parseri osa. SAX parser on kiirem ja effektiivsem, sest ta ei loe kogu dokumenti mällu vaid genereerib sündmuse iga osa andmete kohta mis sisse loetakse.

ddocSaxReadSignedDocFromFile()

Loeb allkirjastatud dokumendi antud failist

- ppSigDoc – puffer allokeeritud dokumendi info jaoks
- szFileName – sisendfaili nimi
- checkFileDigest – (0/1) kas kontrollida viidatavate andmefailide räsikoode.
- lMaxDfLen – suurim algandmefaili suurus mille puhul tema andmedi veel hoitakse mälus.

```
int readSignedDoc(SignedDoc** ppSigDoc, const char* szFileName, int checkFileDigest, int lMaxDfLen);
```

ddocSaxReadSignedDocFromMemory()

Loeb allkirjastatud dokumendi mälupuhvrist

- ppSigDoc – puffer allokeeritud dokumendi info jaoks
- pData – sisendandmed
- len – sisendandmete pikkus baitides.
- lMaxDfLen – suurim algandmefaili suurus mille puhul tema andmedi veel hoitakse mälus.

```
int ddocSaxReadSignedDocFromMemory(SignedDoc** ppSigDoc,

const void* pData, int len, long lMaxDFLen);
```

ddocSaxExtractDataFile()

Loeb allkirjastatud dokumendi antud failist ja salvestab soovitud algandmefaili antud uude faili

- pSigDoc – Allkirjastatud digidoc dokumendi objekt
- szFileName – sisendfaili nimi
- szDataFileName – väljundfaili nimi
- szDocId – soovitud algandmefaili tunnus
- szCharset - soovitav väljundi kooditabel

Funktsioon tagastab ERR_OK või veakoodi.

```
int ddocSaxExtractDataFile(SignedDoc* pSigDoc,

const char* szFileName, const char* szDataFileName,

const char* szDocId, const char* szCharset);
```

Allkirjastatud dokumendi lugemine XML-Reader API abil.

Selle kategooria funktsioonide abil saab allkirjastatud dokumente failist lugeda ja eraldi faili salvestada. Antud moodul kasutab libxml2 teegi XML-Reader parseri osa. XML-Reader parser võimaldab kasutada ka X-Path -i mida oleks vaja osade XAdES implementatsioonide loodud failide lugemiseks.

ddocXRdrReadSignedDocFromFile()

Loeb allkirjastatud dokumendi antud failist

- ppSigDoc – puffer allokeeritud dokumendi info jaoks
- szFileName – sisendfaili nimi
- lMaxDfLen – suurim algandmefaili suurus mille puhul tema andmedi veel hoitakse mälus.

```
int ddocXRdrReadSignedDocFromFile(const char* szFileName, SignedDoc**
ppSigDoc, long lMaxDFLen);
```

ddocXRdrReadSignedDocFromMemory()

Loeb allkirjastatud dokumendi antud failist

- ppSigDoc – puhver allokeeritud dokumendi info jaoks
- szXml – sisendandmed
- xmlLen – sisendandmete pikkus baitides.
- lMaxDfLen – suurim algandmefaili suurus mille puhul tema andmedi veel hoitakse mälus.

```
int ddocXRdrReadSignedDocFromMemory(const char* szXml,

int xmlLen, SignedDoc** pSigDoc, long lMaxDFLen);
```

ddocXRdrExtractDataFile()

Loeb allkirjastatud dokumendi antud failist ja salvestab soovitud algandmefaili antud uude faili

- pSigDoc – Allkirjastatud digidoc dokumendi objekt
- szFileName – sisendfaili nimi
- szDataFileName – väljundfaili nimi
- szDocId – soovitud algandmefaili tunnus
- szCharset - soovitav väljundi kooditabel

Funktsioon tagastab ERR_OK või veakoodi.

```
int ddocXRdrExtractDataFile(SignedDoc* pSigDoc,

const char* szFileName, const char* szDataFileName,

const char* szDocId, const char* szCharset);
```

ddocXRdrGetDataFile()

Loeb allkirjastatud dokumendi antud failist ja tagastab selle andmed uues mälupuhvris. Kasutaja peab selle mälu vabastama.

- pSigDoc – Allkirjastatud digidoc dokumendi objekt
- szFileName – sisendfaili nimi
- szDataFileName – väljundfaili nimi
- szDocId – soovitud algandmefaili tunnus
- pBuf – mälupuhvriobjekt (andmed ja pikkus)

Funktsioon tagastab ERR_OK või veakoodi.

```
int ddocXRdrGetDataFile(SignedDoc* pSigDoc, const char* szFileName,

const char* szDocId, DigiDocMemBuf* pBuf);
```

PKCS11 funktsioonid

Selle kategooria funktsioonide abil saab lugeda kasutada kiipkaarti nii Windows kui Linux/UNIX keskkonnas. Võimalik on kasutada kas Eesti Ühispanga loodud voi Marie Fischer ja Martin Paljaku täiendustega OpenSC PKCS#11 ohjurprogrammi.

initPKCS11Library()

Laeb mällu ja initsialiseerib PKCS#11 ohjurprogrammi.

- libName – PKCS#11 ohjurprogrammi nimi (s.o. .dll või .so nimi)

Funktsioon tagastab PKCS#11 teegi handle mida on vaja edasises töös.

```
LIBHANDLE initPKCS11Library(const char* libName);
```

closePKCS11Library()

Suleb teegi- ja/või kaardisessiooni.

- pLibrary – PKCS#11 teegi handle
- hSession – kaardisessiooni handle

Funktsioon tagastab PKCS#11 teegi handle mida on vaja edasises töös.

```
void closePKCS11Library(LIBHANDLE pLibrary, CK_SESSION_HANDLE
hSession);
```

GetSlotIds()

Tagastab PKCS#11 „slottide“ tunnuste loetelu.

- pSlotids – PKCS#11 „slottide“ tunnuste jada aadress.
- pLen – jada algne pikkus / mahutavus. Siin tagastatakse ka leitud tunnuste arv.

Funktsioon tagastab PKCS#11 taseme veakoodi või CKR_OK.

```
CK_RV GetSlotIds(CK_SLOT_ID_PTR pSlotids, CK_ULONG_PTR pLen);
```

GetTokenInfo()

Tagastab soovitud PKCS#11 „tokeni“ andmed. Üks token vastab ühele võtmepaarile.

- pTokInfo – PKCS#11 „tokeni“ detailandmete puhvri aadress.
- id – valitud sloti tunnus.

Funktsioon tagastab PKCS#11 taseme veakoodi või CKR_OK.

```
CK_RV GetTokenInfo(CK_TOKEN_INFO_PTR pTokInfo, CK_SLOT_ID id);
```

getDriverInfo()

Tagastab PKCS#11 ohjurprogrammi andmed.

- pInfo – PKCS#11 ohjurprogrammi detailandmete puhvri aadress.

Funktsioon tagastab PKCS#11 taseme veakoodi või CKR_OK.

```
CK_RV getDriverInfo(CK_INFO_PTR pInfo);
```

GetSlotInfo()

Tagastab soovitud PKCS#11 „sloti“ andmed. Üks slot vastab ühele kaardilugejale. OpenSC jagab kaartidel leitud võtmepaarid (tokenid) sedasi virtuaalsetesse slottidesse et igas slotis on täpselt üks token.

- pSlotInfo – PKCS#11 „sloti“ detailandmete puhvri aadress.
- id – valitud sloti tunnus.

Funktsioon tagastab PKCS#11 taseme veakoodi või CKR_OK.

```
CK_RV GetSlotInfo(CK_SLOT_INFO_PTR pSlotInfo, CK_SLOT_ID id);
```

loadAndTestDriver()

Laeb PKCS#11 ohjurprogrammi, loeb slottide ja tokenite andmed ja kontrollib vajaliku sloti olemasolu.

- driver – PKCS#11 ohjruprogrammi faili nimi.
- pLibrary – puhver teegi handle salvestamiseks.
- slotids – slottide jada aadress
- slots – slottide jada mahutavus
- slot – soovitud sloti järjekorranumbr.

Funktsioon tagastab veakoodi või ERR_OK.

```
int loadAndTestDriver(const char* driver, LIBHANDLE* pLibrary,
CK_SLOT_ID* slotids, int slots, CK_ULONG slot);
```

calculateSignatureWithEstID()

Arvutab ID kaardiga RSA+SHA allkirja väärtuse ja salvestab antud allkirja objektis.

- pSigDoc – digidoc dokumendi objekt.
- pSigInfo – uue allkirja objekt. Siia salvestatakse ka arvutatud allkirja väärtus.
- slot – allkirjastamiseks kasutatava sloti järjekorranumber.
- passwd – allkirjastamiseks vajalik PIN kood.

Funktsioon tagastab veakoodi või ERR_OK.

```
int calculateSignatureWithEstID(SignedDoc* pSigDoc, SignatureInfo*
pSigInfo, int slot, const char* passwd);
```

findUsersCertificate()

Loeb kasutada sertifikaadi kaardilt.

- slot – PKCS#11 sloti number (näiteks 0)
- ppCert – uue sertifikaadi osuti puhvri aadress. Kasutaja peab vabastama teegi poolt allokeeritud sertifikaadi obejekti.

Funktsioon tagastab veakoodi või ERR_OK.

```
int findUsersCertificate(int slot, X509** ppCert);
```

Konfiguratsioonifaili funktsioonid

Selle kategooria funktsioonide abil saab lugeda ja kirjutada konfiguratsioonifaile, ning kasutada lihtsustatud allkirjastamisfunktsioone, mille puhul osa sageli korduvatest väärtustest loetakse konfiguratsioonifailist. DigiDoc teek kasutab Linux/UNIX keskkonnas kahte eri konfiguratsioonifaili:

- globaalne - /etc/digidoc.conf
- kasutaja oma / privaatne - ~/.digidoc.conf

Windows keskkonnas kasutatakse vaid ühte konfiguratsioonifaili – c:\programs\digidoclib\digidoc.ini, mis tulevikus tõenäoliselt asendatakse registry kasutamisega. Globaalse ja privaatse konfiguratsioonifaili kasutamisel mõjub globaalne fail kõigile antud arvuti kasutajatele aga privaatne ainult antud kasutajale. Mõlemas failis võib kasutada samu kirjeid aga privaatses failis salvestatud valikud võetakse kasutusele alati kui nad on olemas ja globaalse faili kirjeid arvestatakse ainult siis kui privaatses failis sellist kirjet ei olnud.

initConfigStore()

Initsialiseerib API ja loeb konfiguratsioonifailidest andmed.

- szConfigFile – võimaldab edastada konfiguratsioonifaili täielikku nime ja teeonda. Kui edastada NULL siis kasutatakse vaikeväärtust.

Funktsioon tagastab veakoodi või ERR_OK.

```
int initConfigStore();
```

cleanupConfigStore()

Eemaldab mälust konfiguratsioonifailidest loetud andmed.

Funktsioon tagastab veakoodi või ERR_OK.

```
int cleanupConfigStore();
```

addConfigItem()

Lisab uue konfiguratsiooni kirje. Seejuures faili otseselt ei salvestata, selleks on oma funktsioon.

- Key – kirje tunnus
- value – kirje väärtus
- type – kirje tüüp: ITEM_TYPE_GLOBAL või ITEM_TYPE_PRIVATE
- status – kirje staatus: ITEM_STATUS_OK või ITEM_STATUS_MODIFIED. Enne salvestamist peaks olema ITEM_STATUS_MODIFIED.

Funktsioon tagastab veakoodi või ERR_OK.

```
int addConfigItem(const char* key, const char* value, int type, int status);
```

createOrReplacePrivateConfigItem()

Lisab uue konfiguratsiooni kirje kasutaja privaatsesse faili. Asendab olemasoleva kirje väärtuse kui selline juba eksisteerib.

- Key – kirje tunnus
- value – kirje väärtus

Funktsioon tagastab veakoodi või ERR_OK.

```
int createOrReplacePrivateConfigItem(const char* key, const char* value);
```

ConfigItem_lookup()

Otsib mingi tunnusega kirje väärtuse alguses privaatsest ja siis globaalsest failist.

- Key – kirje tunnus

Funktsioon tagastab vastava tunnusega kirje väärtuse või NULL kui sellist ei leidunud.

```
const char* ConfigItem_lookup(const char* key);
```

ConfigItem_lookup_int()

Otsib mingi tunnusega numbrilise kirje väärtuse alguses privaatsest ja siis globaalsest failist.

- Key – kirje tunnus
- defValue - vaikeväärtus

Funktsioon tagastab vastava tunnusega kirje väärtuse või vaikeväärtuse kui sellist ei leidunud.

```
int ConfigItem_lookup_int(const char* key, int defValue);
```

ConfigItem_lookup_bool()

Otsib mingi tunnusega loogilise kirje väärtuse alguses privaatsest ja siis globaalsest failist.

- Key – kirje tunnus
- defValue - vaikeväärtus

Funktsioon tagastab vastava tunnusega kirje väärtuse või vaikeväärtuse kui sellist ei leidunud.

```
int ConfigItem_lookup_bool(const char* key, int defValue);
```

ConfigItem_lookup_str()

Otsib mingi tunnusega kirje väärtuse alguses privaatsest ja siis globaalsest failist.

- Key – kirje tunnus
- defValue - vaikeväärtus

Funktsioon tagastab vastava tunnusega kirje väärtuse või vaikeväärtuse kui sellist ei leidunud.

```
const char* ConfigItem_lookup_bool(const char* key, const char* defValue);
```

writePrivateConfigFile()

Kirjutab kirjetele tehtud muudatused privaatsesse konfiguratsioonifaili.

Funktsioon tagastab veakoodi või ERR_OK.

```
int writePrivateConfigFile();
```

notarizeSignature()

Hangib allkirjale kehtivuskinnituse kasutades konfiguratsioonifailides toodud sertifikaate ja valides OCSP responderi sertifikaadi vastavalt responderilt saadud vastusele.

- pSigDoc – digidoc objekt
- pSigInfo – allkiri, millele lisada kehtivuskinnitus.

Funktsioon tagastab veakoodi või ERR_OK.

```
int notarizeSignature(SignedDoc* pSigDoc, SignatureInfo* pSigInfo);
```

signDocument()

Lisab dokumendile allkirja ja hangib ka kehtivuskinnituse.

- pSigDoc – digidoc objekt

- ppSigInfo – aadress uue allkirja osuti jaoks.
- pin – allkirjastamiseks vajalik PIN kood
- manifest – allkirjastaja manifest / roll
- city – allkirjastaja aadress: Linn
- state – allkirjastaja aadress: maakond
- zip – allkirjastaja aadress: postiindeks
- country – allkirjastaja aadress: Maa

Funktsioon tagastab veakoodi või ERR_OK.

```
int signDocument(SignedDoc* pSigDoc, SignatureInfo** ppSigInfo,

const char* pin, const char* manifest,

const char* city, const char* state,

const char* zip, const char* country);
```

verifyNotary()

Kontrollib kehtivuskinnituse allkirja.

- pSigDoc – digidoc objekt
- pNotInfo – kehtivuskinnitus.

Funktsioon tagastab veakoodi või ERR_OK.

```
int verifyNotary(SignedDoc* pSigDoc, NotaryInfo* pNotInfo);
```

verifySignatureAndNotary()

Kontrollib allkirja ja kehtivuskinnituse andmeid.

- pSigDoc – digidoc objekt
- pSigInfo – allkirja objekt.
- szFileName – sisendandmete fail / digidoc dokument.

Funktsioon tagastab veakoodi või ERR_OK.

```
int verifySignatureAndNotary(SignedDoc* pSigDoc, SignatureInfo*
pSigInfo, const char* szFileName);
```

Sertifikaatide funktsioonid

Selle kategooria funktsioonide abil saab lugeda mitmesuguseid allkirjastaja ja kehtivuskinnituse sertifikaadi andmeid.

getSignCertData()

Tagastab allkirjastaja sertifikaadi andmed (X509*)

- pSigInfo – allkirja info

```
void* getSignCertData(const SignatureInfo* pSignInfo);
```

findCertificate()

Tagastab sertifikaadi vastavalt otsingu kriteeriumitele

- certSearch - sertifikaadi otsingu kriteeriumite struktuur

```
X509* findCertificate(const CertSearch * certSearch);
```

getNotCertData()

Tagastab kehtivuskinnituse sertifikaadi andmed (X509*)

- pNotInfo – kehtivuskinnituse info

```
void* getNotCertData(const NotaryInfo* pNotInfo);
```

getCertIssuerName()

Tagastab sertifikaadi väljastaja nime (DN)

- cert – sertifikaadi info (X509*)
- buf – puffer väljastaja nime jaoks
- buflen – puffri pikkuse aadress.

```
int getCertIssuerName(void* cert, char* buf, int* buflen);
```

getCertSubjectName()

Tagastab sertifikaadi omaniku nime (DN)

- cert – sertifikaadi info (X509*)
- buf – puffer omaniku nime jaoks
- buflen – puffri pikkuse aadress.

```
int getCertSubjectName(void* cert, char* buf, int* buflen);
```

getCertSerialNumber()

Tagastab sertifikaadi numbri

- cert – sertifikaadi info (X509*)
- nr – puffer sertifikaadi numbri jaoks

```
int getCertSerialNumber(void* cert, int* nr);
```

GetCertSerialNumber()

Tagastab sertifikaadi numbri

- certfile – sertifikaadi faili nimi (PEM formaadis)

```
long GetCertSerialNumber(const char *certfile);
```

getCertNotBefore()

Tagastab sertifikaadi kehtivuse alguskuupäeva

- pSigDoc - viide kasutatavale DigiDoc'ile

- cert – sertifikaadi info (X509*)
- timestamp - sertifikaadi kehtivuse alguskuupäev

```
int getCertNotBefore(const SignedDoc* pSigDoc, void* cert, char*
timestamp);
```

getCertNotAfter()

Tagastab sertifikaadi kehtivuse lõppkuupäeva

- pSigDoc - viide kasutatavale DigiDoc'ile

- cert – sertifikaadi info (X509*)
- timestamp - sertifikaadi kehtivuse lõppkuupäev

```
int getCertNotAfter(const SignedDoc* pSigDoc, void* cert, char*
timestamp);
```

saveCert()

Salvestab sertifikaadi faili.

- cert – sertifikaadi info (X509*)
- szFileName – väljudnfaili nimi
- nFormat - väljudnfaili format (FILE_FORMAT_PEM või FILE_FORMAT_ASN1)

```
int saveCert(void* cert, const char* szFileName, int nFormat);
```

decodeCert()

Dekodeerib andtud PEM formaadis baitid X509 sertifikaadi struktuuriks;

- pemData - PEM vormigus sertifikaat

```
void* decodeCert(const char* pemData);
```

encodeCert()

Kodeerib X509 sertifikaadi struktuuri binaarkujule;

- x509 - X509-vormigus sertifikaat
- encodedCert - viit, kuhu kirjutatakse tulemus
- encodedCertLen - viit, kuhu kirjutatakse tulemuse pikkus

```
void encodeCert(const X509* x509, char * encodedCert, int*
encodedCertLen);
```

CertSearchStore_new()

Loob uue CertSearchStore struktuuri, kasutamist saab vaadata näitest "Kuidas otsida sertifikaate DigiDoc'i funktsioonides".

```
CertSearchStore* CertSearchStore_new();
```

CertSearchStore_free()

Kustutab CertSearchStore objekti ja vabastab mälu

```
void CertSearchStore_free(CertSearchStore* certSearchStore);
```

CertSearch_new()

Loob uue CertSearch struktuuri, kasutamist saab vaadata näitest "Kuidas otsida sertifikaate DigiDoc'i funktsioonides".

```
CertSearch* CertSearch_new();
```

CertSearch_free()

Kustutab CertSearch struktuuri ja alamstruktuurid ning vabastab mälu.

```
void CertSearch_free(CertSearch* certSearch);
```

CertList_free ()

Vabastab sertifikaatide nimistu elemendid ning nende sisu.

- pListStart - CertItem viit, mis osutab sertfikaatide nimistule.

```
void CertList_free(CertItem* pListStart);
```

readCertPolicies()

Loeb sertifikaadi ja allkirjastamise reeglid antud sertifikaadist.

- pX509 - sertifikaadi viit.
- pPolicies - reeglite massiivi osuti aadress. Teek allokeerib malu selle massivi jaoks ja salvestab etteantud aadressil allokeeritud malu aadressi. Kasutaja peab selle massiivi hiljem ise vabastama.
- nPols - allokeeritud reeglite arvu osuti.

PolicyIdentifiers_free()

Vabastab allokeeritud reeglite massiivi kasutatud mälu.

- pPolicies - reeglite massiivi osuti.
- nPols - allokeeritud reeglite arv.

isCompanyCPSPolicy()

Kontrollib, kas antud sertifikaadi kasutamise reegel on tüübist "asutuse sertifikaadi kasutamise poliitika" ehk siis kas see sertifikaat on asutuse sertifikaat.

- pPolicy - sertifikaadi kasutamise reegli osuti.

Allkirja kontrolli funktsioonid

Selle kategooria funktsioonide abil saab kontrollida allkirja.

compareByteArrays()

Võrdleb kahte baidijada. Kasutatakse räside kontrollimisel.

- dig1 – esimese räsi väärtus
- len1 - esimese räsi pikkus
- dig2 – teise räsi väärtus
- len2 - teise räsi pikkus
- Tagastab 0 kui räsid on võrdsed

```
int compareByteArrays(const byte* dig1,int len1,const byte* dig2, int len2);
```

verifySigDocDigest()

Kontrollib soovitud algandmefaili räsikoodi antud allkirjas ja tagastab ERR_OK kui räsi on õige

- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info
- pDocInfo - algandmefaili allkirjastatud attribuutide info
- szFileName – algandmefaili nimi DETACHED tüübi jaoks.
- szDataFile – algne allkirjastatud dokumendi fail puhta XML algkuju lugemiseks 1.0 versioonis failides, kus XML-i ei kanoniseeritud.

```
int verifySigDocDigest(const SignedDoc* pSigDoc,const SignatureInfo* pSigInfo,const DocInfo* pDocInfo,const char* szFileName, const char* szDataFile);
```

verifySigDocMimeDigest()

Kontrollib soovitud algandmefaili andmetüübi räsikoodi antud allkirjas ja tagastab 0 kui räsi on õige

- pSigDoc – allkirjastatud dokumendi info

- pSigInfo – allkirja info
- pDocInfo - algandmefaili allkirjastatud attribuutide info

```
int verifySigDocMimeDigest(const SignedDoc* pSigDoc, const
SignatureInfo* pSigInfo, const DocInfo* pDocInfo);
```

verifySigDocSigPropDigest()

Kontrollib allkirja allkirjastatud omaduste räsikoodi ja tagastab 0 kui räsi on õige

- pSigDoc – allkirjastatud dokumendi info

- pSigInfo – allkirja info
- szDataFile – algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.) Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)

```
int verifySigDocSigPropDigest(const SignedDoc* pSigDoc,const
SignatureInfo* pSigInfo,const char* szDataFile);
```

verifySignatureInfo()

Kontrollib seda allkirja ülalkirjaldatud funktsioonide abil

- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info
- signerCA – allkirjastaja CA sertifikaadi faili nimi (PEM)
- szDataFile - algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.)Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)
- bUseCA - 1=kontrolli ka antud CA sertifikaadi abil allkirjastaja sertifikaadi kuuluvust tuntud CA-le, 0=ära kontrolli CA abil.

```
int verifySignatureInfo(const SignedDoc* pSigDoc,const SignatureInfo*
pSigInfo,const char* signerCA, const char* szDataFile);
```

verifySignatureInfoCERT()

Kontrollib seda allkirja ülalkirjaldatud funktsioonide abil, eelmisega võrreldes on erinevuseks see, et sertifikaati faili asemel on parameetriks sertifikaadi struktuuri.

- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info
- signerCA – allkirjastaja CA sertifikaat
- szDataFile - algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.)Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)
- bUseCA - 1=kontrolli ka antud CA sertifikaadi abil allkirjastaja sertifikaadi kuuluvust tuntud CA-le, 0=ära kontrolli CA abil.

```
int verifySignatureInfoCERT(const SignedDoc* pSigDoc, const
SignatureInfo* pSigInfo, const void* signerCACert, const char*
szDataFile);
```

verifySignatureInfo_ByCertStore()

Kontrollib seda allkirja ülalkirjaldatud funktsioonide abil, eelmistega võrreldes on erinevuseks see, et allkirjastaja sertifikaat loetakse dokumendist ning ahel leitakse MS sertifikaadihoidlast.

- pSigDoc – allkirjastatud dokumendi info
- pSigInfo – allkirja info
- szDataFile - algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.) Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)

```
int verifySignatureInfo_ByCertStore(const SignedDoc* pSigDoc, const
SignatureInfo* pSigInfo, const char* szDataFile);
```

verifySigDoc()

Kontrollib kogu allkirjastatud dokumenti

- pSigDoc – allkirjastatud dokumendi info
- signerCA – allkirjastaja CA sertifikaadi faili nimi (PEM)
- notaryCA – kehtivuskinnituse andja CA sertifikaadi faili nimi
- rootCA – juur CA sertifikaadi faili nimi
- caPath – sertifikaatide kataloogi nimi

- notCert – kehtivuskinnituse andja sertifikaadi faili nimi
- szDataFile - algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.) Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)
- bUseCA - 1=kontrolli ka antud CA sertifikaadi abil allkirjastaja sertifikaadi kuuluvust tuntud CA-le, 0=ära kontrolli CA abil.

```
int verifySigDoc(const SignedDoc* pSigDoc, const char* signerCA,const
char* notaryCA, const char* rootCA,const char* caPath, const char*
notCert,const char* szDataFile);
```

verifySigDocCERT()

Teeb sama mis eelmine funktsioon verifySigDoc() ehk kontrollib kogu allkirjastatud dokumenti, ainsaks erinevuseks on see, et sertifikaadi failinimede asemel on parameetriteks sertifikaatide struktuurid ise.

- pSigDoc – allkirjastatud dokumendi info
- signerCA – allkirjastaja CA sertifikaat
- notaryCA – kehtivuskinnituse andja CA sertifikaat
- rootCA – juur CA sertifikaat
- caPath – sertifikaatide kataloogi nimi võib olla null
- notCert – kehtivuskinnituse andja sertifikaat

- szDataFile - algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.) Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)
- bUseCA - 1=kontrolli ka antud CA sertifikaadi abil allkirjastaja sertifikaadi kuuluvust tuntud CA-le, 0=ära kontrolli CA abil.

```
int verifySigDocCERT(const SignedDoc* pSigDoc, const void* signerCA,
const void* notaryCA, const void* rootCA, const char* caPath, const
void* notCert, const char* szDataFile);
```

verifySigDoc_ByCertStore ()

Teeb sama mis teised verifySigDocXxx() funktsioonid, aga loeb sertifikaadid dokumendi seest ning leiab ahela MS sertifikaadihoidla abil.

- pSigDoc – allkirjastatud dokumendi info

- szDataFile - algne allkirjastatud dokumendi fail allkirjastatud omaduste lugemiseks nende esialgses formaadis. Kasulik siis kui fail on koostatud teise teegi poolt, mis formateerib XML-i natuke erinevalt (reavahetused vms.) Kui selle parameetri väärtuseks panna NULL, siis koostab ise allkirjastatud omaduste XML kuju mälus olevate andmete pealt. See on kiirem kuid toimib vaid siis kui algne fail on tehtud just selle teegiga (või täpselt sama formaati kasutavaga)

```
int verifySigDoc_ByCertStore(const SignedDoc* pSigDoc, const char*
szDataFile)
```

isCertValid()

kontrollib kas antud sertifikaat on kehtiv. Seda oma algus- ja lõppkuupäeva alusel!!!

- cert – sertifikaadi andmed (X509*)

```
int isCertValid(void* cert);
```

isCertSignedBy()

kontrollib kas antud sertifikaat on allkirjastatud antud CA sertifikaadi poolt

- cert – sertifikaadi andmed (X509*)
- cafile – CA sertifikaadi faili nimi

```
int isCertSignedBy(void* cert, const char* cafile);
```

isCertSignedByCERT()

kontrollib kas antud sertifikaat on allkirjastatud antud CA sertifikaadi poolt,ainsaks erinevuseks on see, et sertifikaatide failinimede asemel on parameetriteks sertifikaatide struktuurid ise.

- cert – uuritav sertifikaat
- cafile – CA sertifikaat

```
int isCertSignedByCERT(const void* cert, const void* caCert);
```

verifySigCert()

Kontrollib, kas allkirjastaja sertifikaadi andmed (number ja räsi) vastavad allkirjastatud omadustes toodud andmetele

- pSigInfo – allkirja info

```
int verifySigCert(const SignatureInfo* pSigInfo);
```

verifyNotaryInfo()

Kontrollib kehtivuskinnituse allkirja

- pSigDoc – allkirjastatud dokumendi info
- pNotInfo - kehtivuskinnituse info
- fileEstEidSK – allkirjastaja ja kehtivuskinnituse CA sertifikaadi faili nimi (PEM)
- fileJuurSK – juur CA sertifikaadi faili nimi
- CApath – sertifikaatide kataloogi nimi
- notCertFile – kehtivuskinnituse andja sertifikaadi faili nimi

```
int verifyNotaryInfo(const SignedDoc* pSigDoc, const NotaryInfo*
pNotInfo,const char *fileJuurSK, const char *fileEstEidSK,const char
*CApath, const char* notCertFile);
```

verifyNotaryInfoCERT()

Kontrollib kehtivuskinnituse allkirja,ainsaks erinevuseks on see, et sertifikaatide failinimede asemel on parameetriteks sertifikaatide struktuurid ise.

- pSigDoc – allkirjastatud dokumendi info
- pNotInfo - kehtivuskinnituse info
- certEstEidSK – allkirjastaja ja kehtivuskinnituse CA sertifikaat
- certJuurSK – juur CA sertifikaat
- CApath – sertifikaatide kataloogi nimi
- notCertFile – kehtivuskinnituse andja sertifikaat

```
int verifyNotaryInfoCERT(const SignedDoc* pSigDoc, const NotaryInfo*
pNotInfo, const void *certJuurSK, const void *certEstEidSK,const char
*CApath, const void* notCert);
```

verifyNotaryInfo_ByCertStore()

Kontrollib kehtivuskinnituse allkirja, ainsaks erinevuseks eelmisest on see, et funktsioon loeb sertifikaadid dokumendi seest ning leiab ahela MS sertifikaadihoidla abil.

- pSigDoc – allkirjastatud dokumendi info
- pNotInfo - kehtivuskinnituse info

```
int verifyNotaryInfo_ByCertStore(const SignedDoc* pSigDoc, const
NotaryInfo* pNotInfo);
```

verifyNotCert ()

Kontrollib, kas kehtivuskinnituse sertifikaadi andmed (number ja räsi) vastavad allkirjastatud omadustes toodud andmetele

- pSigInfo – allkirja info

```
int verifyNotCert(const NotaryInfo* pNotInfo);
```

verifyNotaryDigest()

Kontrollib antud notari info räsikoodi.

- pSigDoc - viide kasutatavale DigiDoc'ile
- pNotInfo - notari info struktuur

```
int verifyNotaryDigest(const SignedDoc* pSigDoc, const NotaryInfo*
pNotInfo);
```

writeOCSPRequest()

Koostab OCSP formaadis kehtivuskinnituse taotluse ja salvestab at etteantud faili

- signerCertFile – allkirjastaja sertifikaadi faili nimi
- issuertCertFile - allkirjastaja CA sertifikaadi faili nimi
- nonce - allkirja räsi
- nlen - allkirja räsi pikkus
- szOutputFile – väljundfaili nimi

```
int writeOCSPRequest(const char* signerCertFile, const char*
issuertCertFile,byte* nonce, int nlen, const char* szOutputFile);
```

getConfirmation()

Loob (ja allkirjastab) notari (OCSP) päringu, saadab selle OCSP responderile, parsib vastuse ning lisab vastuse antud SignedDoc struktuuri.

Selle funktsiooni kasutamist saab vaadata näitest "Kuidas lisada kehtivuskinnitust ?"

- pSigDoc - SignedDoc struktuur

- pSigInfo - SignatureInfo struktuur, mille allkirjasta koha küsitakse kehtivus kinnitust.
- notaryCert - notari sertifikaat
- pkcs12FileName – pääsutõendi failinimi
- pkcs12Password – pääsutõendi paroolifraas
- signCert - notari päringu allkirjasta sertifikaat
- notaryURL - notaryi (OCSP responderi) URL
-
- proxyHost - kui notari URL'I kätte saamiseks on vajalik proksi siis selle nimi
- proxyPort - proksi pordi number

```
int getConfirmation(SignedDoc* pSigDoc, SignatureInfo* pSigInfo,

const X509** caCerts, const X509* pNotCert,

char* pkcs12FileName, char* pkcs12Password,

char* notaryURL, char* proxyHost, char* proxyPort)
```

calculateNotaryInfoDigest()

Arvutab räsikoodi üle notari info struktuuri, tagastab ERR_OK või veakoodi.

- pNotInfo - notari info struktuur
- digBuf - väljund puhver räsikoodile
- digLen - väljund puhveri pikkus (siia kirjutatakse räsi pikkus)

```
int calculateNotaryInfoDigest(const NotaryInfo* pNotInfo, byte*
digBuf, int* digLen);
```

getSignerCode()

Loeb antud allkirja andnud isiku ID koodi

- pSigInfo - uuritava allkirja info struktuur
- buf - eelnevalt allokeeritud väljund puhver ID koodile

```
int getSignerCode(const SignatureInfo* pSigInfo, char* buf);
```

getSignerFirstName()

Loeb antud allkirja andnud isku eesnime

- pSigInfo - uuritava allkirja info struktuur

- buf - eelnevalt allokeeritud väljund puhver eesnime jaoks

```
int getSignerFirstName(const SignatureInfo* pSigInfo, char* buf);
```

getSignerLastName()

Loeb antud allkirja andnud isku perekonnanime

- pSigInfo - uuritava allkirja info struktuur
- buf - eelnevalt allokeeritud väljund puhver perekonnanime jaoks

```
int getSignerLastName(const SignatureInfo* pSigInfo, char* buf);
```

Krüpteerimise ja dekrüpteerimise funktsioonid

Selle kategooria funktsioonide abil saab luua krüpteeritud faile, mis vastavad XML-ENC standardile, neid lugeda ja derüpteerida. API salvestab sisseloetud krüpteeritud faili andmed C strktuurides ja pakub ka funktsioone iga sellise struktuurielemendi lugemiseks ja muutmiseks. Soovitav oleks kasutada neid funktsioone struktuuri otsese muutmise asemel, sest funktsioonid teevad ka näiteks veakontrolli. Igas krüpteeritud failis on vaid üks <EncryptedData> element mis sisaldab ühe ainsa krüpteeritud andmeelemendi. Sellel elemendil on aga üks või mitu <EncryptedKey> alamelementi, üks iga dokumendi vastuvõtja jaoks. Vastuvõtja on isik, kelle avaliku võtmega on krüpteeritud üks koopia antud dokumendi transpordivõtmest ja kes seega on suuteline seda dokumenti oma ID kaardiga dekrüpteerima.

dencEncryptedData_new()

Loob uue <EncryptedData> objekti mälus.

- ppEncData – aadress kuhu salvestada loodava objekti pointer
- szXmlNs – XML namespace väärtus
- szEncMethod – <EncryptionMethod> alamelemendi väärtus
- szId – antud elemendi ID atribuudi väärtus
- szType – antud elemendi Type atribuudi väärtus
- szMimeType – antud elemendi MimeType atribuudi väärtus
- Tagastab veakoodi või ERR_OK (0)

dencEncryptedData_free()

Kustutab <EncryptedData> objekti ja kõik tema alamobjektid mälust.

- pEncData – kustutatava objekti aadress
- Tagastab veakoodi või ERR_OK (0)

dencEncryptedData_GetId()

Tagastab <EncryptedData> objekti ID atribuudi väärtuse.

- pEncData – objekti aadress
- Tagastab ID atribuudi väärtuse või NULL

dencEncryptedData_GetType()

Tagastab <EncryptedData> objekti Type atribuudi väärtuse.

- pEncData – objekti aadress
- Tagastab Type atribuudi väärtuse või NULL

dencEncryptedData_GetMimeType()

Tagastab <EncryptedData> objekti MimeType atribuudi väärtuse.

- pEncData – objekti aadress
- Tagastab MimeType atribuudi väärtuse või NULL

dencEncryptedData_GetMimeXmlNs()

Tagastab <EncryptedData> objekti xmlns atribuudi väärtuse.

- pEncData – objekti aadress
- Tagastab xmlns atribuudi väärtuse või NULL

dencEncryptedData_GetEncryptionMethod()

Tagastab <EncryptedData> objekti EncryptionMethod alamelemendi väärtuse.

- pEncData – objekti aadress
- Tagastab EncryptionMethod alamelemendi väärtuse või NULL

dencEncryptedData_GetEncryptionPropertiesId()

Tagastab <EncryptedData> objekti <EncryptionProperties> alamelemendi ID atribuudi väärtuse.

- pEncData – objekti aadress
- Tagastab <EncryptionProperties> alamelemendi ID atribuudi väärtuse või NULL

dencEncryptedData_GetEncryptionPropertiesCount()

Tagastab <EncryptedData> objekti <EncryptionProperties> / <EncryptionProperty> alamelementide arvu.

- pEncData – objekti aadress
- Tagastab <EncryptionProperty> alamelementide arvu või -1 vea puhul.

dencEncryptedData_GetEncryptionProperty()

Tagastab <EncryptedData> objekti <EncryptionProperty> alamelemendi.

- pEncData – objekti aadress
- nIdx – alamelemendi index (alates 0 -st)
- Tagastab <EncryptionProperty> alamelemendi aadressi või NULL

dencEncryptedData_GetLastEncryptionProperty()

Tagastab <EncryptedData> objekti viimase <EncryptionProperty> alamelemendi.

- pEncData – objekti aadress
- Tagastab viimase <EncryptionProperty> alamelemendi aadressi või NULL

dencEncryptedData_FindEncryptionPropertyByName()

Tagastab <EncryptedData> objekti sellise <EncryptionProperty> alamelemendi, mille Name atribuudil on soovitud väärtus.

- pEncData – objekti aadress
- name – soovitud alamelemendi Name atribuudi väärtus
- Tagastab <EncryptionProperty> alamelemendi aadressi või NULL

dencEncryptedData_GetEncryptedKeyCount()

Tagastab <EncryptedData> objekti <EncryptedKey> alamelementide arvu.

- pEncData – objekti aadress
- Tagastab <EncryptedKey> alamelementide arvu või -1 vea puhul.

dencEncryptedData_GetEncryptedKey()

Tagastab <EncryptedData> objekti <EncryptedKey> alamelemendi.

- pEncData – objekti aadress
- nIdx – alamelemendi index (alates 0 -st)
- Tagastab <EncryptedKey> alamelemendi aadressi või NULL

dencEncryptedData_FindEncryptedKeyByRecipient()

Tagastab <EncryptedData> objekti sellise <EncryptedKey> alamelemendi mille Recipient atribuudil on toodud väärtus

- pEncData – objekti aadress
- recipient – soovitud alamelemendi Recipient atribuudi väärtus.
- Tagastab <EncryptedKey> alamelemendi aadressi või NULL

dencEncryptedData_FindEncryptedKeyByCN()

Tagastab <EncryptedData> objekti sellise <EncryptedKey> alamelemendi mille sertifikaadi omaniku DN välja CN alamväljas on toodud väärtus

- pEncData – objekti aadress
- cn – soovitud alamelemendi sertifikaadi omaniku DN välja CN alamvälja väärtus.
- Tagastab <EncryptedKey> alamelemendi aadressi või NULL

dencEncryptedData_GetLastEncrypted()

Tagastab <EncryptedData> objekti viimase <EncryptedKey> alamelemendi.

- pEncData – objekti aadress
- Tagastab <EncryptedKey> alamelemendi aadressi või NULL

dencEncryptedData_GetEncryptedData()

Tagastab <EncryptedData> objektis olevate krüpteeritud andmete mälupuhvri.

- pEncData – objekti aadress
- ppBuf – aadress kuhu salvestada krüpteeritud andmete mälupuhvri pointer.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_GetEncryptedDataStatus()

Tagastab <EncryptedData> objektis olevate krüpteeritud andmete staatuse.

- pEncData – objekti aadress
- ppBuf – aadress kuhu salvestada krüpteeritud andmete mälupuhvri pointer.
- Tagastab krüpteeritud andmete staatuse

dencEncryptedData_SetId()

Omistab <EncryptedData> objekti ID atribuudile uue väärtuse.

- pEncData – objekti aadress
- value – omistatav väärtus
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_SetType()

Omistab <EncryptedData> objekti Type atribuudile uue väärtuse.

- pEncData – objekti aadress
- value – omistatav väärtus
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_SetMimeType()

Omistab <EncryptedData> objekti MimeType atribuudile uue väärtuse.

- pEncData – objekti aadress
- value – omistatav väärtus
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_SetXmlNs()

Omistab <EncryptedData> objekti xmlns atribuudile uue väärtuse.

- pEncData – objekti aadress
- value – omistatav väärtus
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_SetEncryptionMethod()

Omistab <EncryptedData> objekti <EncryptionMethod> alamelemendile uue väärtuse.

- pEncData – objekti aadress
- value – omistatav väärtus
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_AppendData()

Omistab või lisab <EncryptedData> objektis hoitud, hetkel veel krüpteerimata, andmetele uue andmebloki. Selle funktsioon abil lisatakse <EncryptedData> objektile andmed, mida siis järgmistes sammudes krüpteerima hakatakse.

- pEncData – objekti aadress
- data – uus andmeblokk
- len – andmete pikkus baitides. Kasuta -1 kui lisad 0 -ga lõpetatud stringi.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_SetEncryptionPropertiesId()

Omistab <EncryptedData> objekti <EncryptionProperties> alamelemendi ID atribuudile uue väärtuse.

- pEncData – objekti aadress
- value – omistatav väärtus
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_DeleteEncryptionProperty()

Kustutab <EncryptedData> objekti <EncryptionProperty> alamelemendi.

- pEncData – objekti aadress
- nIdx – kustutatava alamelemendi index (alates 0 -st)
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_DeleteEncryptedKey()

Kustutab <EncryptedData> objekti <EncryptedKey> alamelemendi.

- pEncData – objekti aadress
- nIdx – kustutatava alamelemendi index (alates 0 -st)
- Tagastab veakoodi või ERR_OK (0).

dencEncryptionProperty_new()

Loob uue <EncryptionProperty> objekti ja lisab ta <EncryptedData> objekti vastavasse loetelusse. Üldiselt kasutame neid alamobjekte mingi doumendi omaduse nagu näiteks faili nimi, originaalsuuruse või mime tüübi salvestamiseks. Sel juhul omistame atribuudile Name salvestata omaduse tunnuse, näiteks „Filename“, ja elemendi sisus salvestame antud väärtuse. On loodud ka funktsioonid Name atribuudi väärtuse järgi sellise objekti otsimiseks.

- pEncData – EncryptedData objekti aadress loodava alamobjekti jaoks
- pEncProperty – aadress kuhu salvestada loodava alamobjekti pointer
- szId – objekti ID atribuudi väärtus (optional)
- szTarget – objekti Target atribuudi väärtus (optional)
- szName – objekti Name atribuudi väärtus (optional)
- szContent – objekti sisu (optional)
- Tagastab veakoodi või ERR_OK (0).

dencEncryptionProperty_free()

Kustutab <EncryptionProperty> objekti.

- pEncProperty – kustutatava objekti aadress
- Tagastab veakoodi või ERR_OK (0).

dencEncryptionProperty_GetId()

Tagastab <EncryptionProperty> objekti ID atribuudi väärtuse.

- pEncProp – objekti aadress
- Tagastab ID atribuudi väärtuse või NULL

dencEncryptionProperty_GetTarget()

Tagastab <EncryptionProperty> objekti Target atribuudi väärtuse.

- pEncProp – objekti aadress
- Tagastab Target atribuudi väärtuse või NULL

dencEncryptionProperty_GetName()

Tagastab <EncryptionProperty> objekti Name atribuudi väärtuse.

- pEncProp – objekti aadress
- Tagastab Name atribuudi väärtuse või NULL

dencEncryptionProperty_GetContent()

Tagastab <EncryptionProperty> objekti sisu.

- pEncProp – objekti aadress
- Tagastab elemendi sisu või NULL

dencEncryptionProperty_SetId()

Omistab <EncryptionProperty> objekti ID atribuudile uue väärtuse.

- pEncProp – objekti aadress
- value – atribuudi uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptionProperty_SetTarget()

Omistab <EncryptionProperty> objekti Target atribuudile uue väärtuse.

- pEncProp – objekti aadress
- value – atribuudi uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptionProperty_SetName()

Omistab <EncryptionProperty> objekti Name atribuudile uue väärtuse.

- pEncProp – objekti aadress
- value – atribuudi uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptionProperty_SetContent()

Omistab <EncryptionProperty> objektile uue sisu.

- pEncProp – objekti aadress
- value – objekti uus sisu
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_new()

Loob uue <EncryptedKey> objekti ja lisab ta <EncryptedData> objekti vastavasse loetelusse. Neid objekte kasutame krüpteeritud dokumendi vastuvõtjate andmete salvestamiseks. Iga vastuvõtja kohta üks selline objekt. Vajalik on vähemalt vastuvõtja sertifikaat, sest sellega krüpteerime tegelike dokumendi andmete krüpteerimiseks kasutatud AES transpordivõtme.

- pEncData – EncryptedData objekti aadress loodava alamobjekti jaoks

- pEncKey – aadress kuhu salvestada loodava alamobjekti pointer
- szEncMethod - <EncryptionMethod> alamobjekti väärtus (vajalik)
- szId – objekti ID atribuudi väärtus (optional)
- szRecipient – objekti Recipient atribuudi väärtus (optional)
- szKeyName – <KeyName> alamobjekti väärtus (optional)
- szCarriedKeyName – <CarriedKeyName> alamobjekti väärtus (optional)
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_free()

Kustutab <EncryptedKey> objekti.

- pEncKey – kustutatava objekti aadress.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_GetId()

Tagastab <EncryptedKey> objekti ID atribuudi väärtuse.

- pEncKey – objekti aadress
- Tagastab ID atribuudi väärtuse või NULL

dencEncryptedKey_GetRecipient()

Tagastab <EncryptedKey> objekti Recipient atribuudi väärtuse.

- pEncKey – objekti aadress
- Tagastab Recipient atribuudi väärtuse või NULL

dencEncryptedKey_GetEncryptionMethod()

Tagastab <EncryptedKey> objekti <EncryptionMethod> alamobjekti väärtuse.

- pEncKey – objekti aadress
- Tagastab <EncryptionMethod> alamobjekti väärtuse või NULL

dencEncryptedKey_GetKeyName()

Tagastab <EncryptedKey> objekti <KeyName> alamobjekti väärtuse.

- pEncKey – objekti aadress
- Tagastab <KeyName> alamobjekti väärtuse või NULL

dencEncryptedKey_GetCarriedKeyName()

Tagastab <EncryptedKey> objekti <CarriedKeyName> alamobjekti väärtuse.

- pEncKey – objekti aadress
- Tagastab <CarriedKeyName> alamobjekti väärtuse või NULL

dencEncryptedKey_GetCertificate()

Tagastab <EncryptedKey> objektis salvestatud vastuvõtja sertifikaadi.

- pEncKey – objekti aadress
- Tagastab vastuvõtja sertifikaadi või NULL

dencEncryptedKey_SetId()

Omistab <EncryptedKey> objekti ID atribuudile uue väärtuse.

- pEncKey – objekti aadress
- value – atribuudi uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_SetRecipient()

Omistab <EncryptedKey> objekti Recipient atribuudile uue väärtuse.

- pEncKey – objekti aadress
- value – atribuudi uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_SetEncryptionMethod()

Omistab <EncryptedKey> objekti <EncryptionMethod> alamobjektile uue väärtuse.

- pEncKey – objekti aadress
- value – alamobjekti uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_SetKeyName()

Omistab <EncryptedKey> objekti <KeyName> alamobjektile uue väärtuse.

- pEncKey – objekti aadress
- value – alamobjekti uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_SetCarriedKeyName()

Omistab <EncryptedKey> objekti <CarriedKeyName> alamobjektile uue väärtuse.

- pEncKey – objekti aadress
- value – alamobjekti uus väärtus.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedKey_SetCertificate()

Omistab <EncryptedKey> objektile vastuvõtja sertifikaadi.

- pEncKey – objekti aadress
- value – vastuvõtja sertifikaadi aadress.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_encryptData()

Krüpteerib dokumendi andmed.

- pEncData – objekti aadress
- nCompressOption – komprimeerimise valik. Võimaliku väärtused on: DENC_COMPRESS_ALLWAYS (komprimeeri), DENC_COMPRESS_NEVER (mitte komprimeerida) või DENC_COMPRESS_BEST_EFFORT (komprimeeri aga kasuta komprimeeritud andmeid vaid siis kui nende maht komprimeerimisel väheneb, vastasel juhul kasuta originaalandmeid)
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_decrypt_withKey()

Dekrüpteerib dokumendi andmed selleks edastatud, juba dekrüpteeritud, AES transpordivõtme abil. Seda funktsiooni kasutatakse siis kui ei soovita kasutada PKCS#11 ohjruprogrammi ja/või ID kaarti AES transpordivõtme dekrüpteerimiseks.

- pEncData – objekti aadress
- tKey – dekrüpteeritud AES transpordivõti.
- KeyLen – transpordivõtme pikkus baitides.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_decryptData()

Dekrüpteerib dokumendi andmed objektis salvestatud, juba dekrüpteeritud, AES transpordivõtme abil. Seda funktsiooni kasutatakse siis kui üks objektis hoitud <EncryptedKey> alamobjektidest on juba dekrüpteeritud vastuvõtja poolt ja seega transpordivõti dekrüpteeritud kujul olemas.

- pEncData – objekti aadress
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_decrypt()

Dekrüpteerib dokumendi andmed objektis salvestatud <EncryptedKey> abil. Vastav ID kaart peab olema sisestatud kaardilugejasse ja PKCS#11 ohjruprogramm installeeritud.

- pEncData – objekti aadress
- pEncKey – valitud vastuvõtja / <EncryptedKey> objekt
- pin – antud vastuvõtja PIN1 kood.
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_compressData()

Komprimeerib dokumendis hoitud andmed. Andmed ei tohi veel olla krüpteeritud.

- pEncData – objekti aadress
- nCompressOption – komprimeerimise valik. Võimaliku väärtused on: DENC_COMPRESS_ALLWAYS (komprimeeri), DENC_COMPRESS_NEVER (mitte komprimeerida) või DENC_COMPRESS_BEST_EFFORT (komprimeeri aga kasuta komprimeeritud andmeid vaid siis kui nende maht komprimeerimisel vähenes, vastasel juhul kasuta originaalandmeid)
- Tagastab veakoodi või ERR_OK (0).

dencEncryptedData_decompressData()

Dekomprimeerib dokumendis hoitud andmed.

- pEncData – objekti aadress
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfo_new()

Loob uue RecvInfo objekti. Neid objekte kasutame krüpteeritud dokumendi vastuvõtjate andmete haldamiseks konfiguratsioonifailis. Iga vastuvõtja kohta üks selline objekt. Vajalik on vähemalt vastuvõtja sertifikaat, sest sellega krüpteerime tegelike dokumendi andmete krüpteerimiseks kasutatud AES transpordivõtme.

- ppRecvInfo – aadress kuhu salvestada loodava alamobjekti pointer
- szId – objekti ID atribuudi väärtus. See atribuut on vajalik vastvõtjate andmete haldamiseks konfiguratsioonifailis ja nende andmete hilisemaks otsinguks.
- szRecipient – objekti Recipient atribuudi väärtus (optional)
- szKeyName – <KeyName> alamobjekti väärtus (optional)
- szCarriedKeyName – <CarriedKeyName> alamobjekti väärtus (optional)
- pCert – vastuvõtja sertifikaat. (nõutud)
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfo_free()

Kustutab RecvInfo objekti.

- pRecvInfo – kustutatava objekti aadress.
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfo_store()

Kirjutab RecvInfo objekti (muudetud?) andmed konfiguratsioonifaili.

- pRecvInfo – kustutatava objekti aadress.
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfo_findById()

Otsi RecvInfo objekti konfiguratsioonifailist tema ID atribuudi väärtuse järgi.

- pConfStore – konfiguratsioonifaili / loetelu objekti aadress. Kasuta NULL väärtust vaikimisi failist lugemiseks. Seda objekti saab kasutada selleks, et näiteks mingite kriteeriumide alusel otsitud ja sellisesse loetelisse salvestatud andmete hulgast nüüd ühe vastuvõtja andmed välja lugeda.
- ppRecvInfo – aadress kuhu salvestada loodud/otsitud objekti osuti.
- szId – ID atribuudi väärtus mille alusel vastuvõtja andmed valida
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfo_delete()

Kustutab RecvInfo objekti konfiguratsioonifailist.

- pRecvInfo – kustutatava objekti osuti.
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfo_findAll()

Loeb kõik RecvInfo objektid konfiguratsioonifailist.

- pRecvInfoList – loetelu RecvInfo objektide hoidmiseks.
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfoList_add()

Lisab ühe RecvInfo objekti loetelusse.

- pRecvInfoList – loetelu RecvInfo objektide hoidmiseks.
- pRecvInfo – lisatava objekti osuti.
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfoList_free()

Vabastab loetelu poolt kasutatud mälu.

- pRecvInfoList – loetelu RecvInfo objektide hoidmiseks.
- Tagastab veakoodi või ERR_OK (0).

dencRecvInfoList_delete()

Kustutab soovitud RecvInfo objekti loetelust.

- pRecvInfoList – loetelu RecvInfo objektide hoidmiseks.
- szId – kustutatava RecvInfo objekti ID atribuut
- Tagastab veakoodi või ERR_OK (0).

dencEncryptFile()

Krüpteerib soovitud sisendfaili ja salvestab väljundfaili. Kasuta seda funktsiooni eriti suurte failide krüpteerimiseks. Antud versioonis ei rakenda see funktsioon veel komprimeerimist.

- pEncData – EncryptedData objekt initsialiseeritud AES transpordivõtme ja vastuvõtjate andmetega.
- szInputFileName – sisendfaili nimi
- szOutputFileName – väljundfaili nimi
- szMimeType – sisendfaili mime tüüp (optional)
- Tagastab veakoodi või ERR_OK (0).

dencGenEncryptedData_toXML()

Genereerib EncryptedData XML vormingu ja tagastab selle DigiDocMemBuf struktuuris. Viimane sisaldab osutit andmetele (pMem) ja andmete mahtu baitides (nLen). Kasutaja peab allokeeritud mälu vabastama DigiDocMemBuf_free() funktsiooniga.

- pEncData – EncryptedData objekt.
- pBuf – mälupuhvri objekt XML tagastamiseks.
- Tagastab veakoodi või ERR_OK (0).

dencGenEncryptedData_writeToFile()

Genereerib EncryptedData XML vormingu ja kirjutab selle soovitud faili.

- pEncData – EncryptedData objekt.
- szFileName – väljundfaili nimi.
- Tagastab veakoodi või ERR_OK (0).

dencSaxReadEncryptedData()

Loeb krüpteeritud faili mällu. Kasuta seda väiksemate ja keskmiste failide jaoks.

- ppEncData – aadress kuhu salvestada loodud EncryptedData objekti osuti.
- szFileName – sisendfaili nimi.
- Tagastab veakoodi või ERR_OK (0).

dencSaxReadDecryptFile()

Loeb krüpteeritud faili osade kaupa sisse, dekrüpteerib ja kirjutab dekrüpteeritud andmed väljundfaili. Seda funktsiooni saab kasutada eriti suurte failide dekrüpteerimiseks. Kasutab PKCS#11 ohjurprogrami dekrüpteerimisel.

- szFileName – sisendfaili nimi.
- szOutputFileName – väljundfaili nimi
- szCertCN – vastuvõtja sertifikaadi omaniku DN välja CN alamväli. Selle alusel valitakse EncryptedKey element mille abil transpordivõtit dekrüpteerda.
- szPin – vastuvõtja ID kaardi PIN1

- Tagastab veakoodi või ERR_OK (0).

dencOrigContent_count()

Loendab EncryptionProperty objektide hulga mis sisaldavad andmeid pakitud ja krüpteeritud digidoc documendi kohta.

- pEncData – EncruptedData objekt [nõutud].
- objektide hulga või - 1

```
EXP_OPTION int dencOrigContent_count(DEncEncryptedData* pEncData);
```

dencOrigContent_add()

Lisab uue EncryptionProperty objekti krüpteeritud digidoc dokumendi andmete salvestamiseks. Seda funktsiooni tuleks kasutada üks kord iga krüpteeritud dokumendi andmefaili kohta. Faili suuruse saab küll edastada stringi kujul kui siia tuleks salvestada otsene baitide arv (ainult numbriline).

- pEncData – EncruptedData objekt [nõutud].
- szOrigContentId – mingi väärtus loodava EncryptionProperty obejkti Id atribuudi jaoks [mitte nõutud]
- szName – andmefaili nimi
- szSize – andmefaili suurus (täpne baitide arv)
- szMime – andmefaili maimi tüüp
- szDfId – andmefaili Id atribuudi väärtus
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencOrigContent_add(DEncEncryptedData* pEncData, const
char* szOrigContentId, const char* szName, const char* szSize, const
char* szMime, const char* szDfId);
```

dencOrigContent_findByIndex()

Loeb soovitud krüpteeritud digidoc documendi andmeid sisaldava EncryptionProperty andmed. Siin kasutatud järjekorranumber ei ole mitte üldine EncryptionProperty objektide järjekorranumber vaid hlmab vaid selliseid objekte mis sisaldavad krüpteeritud digidoc documendi andmeid. Seega algab 0 - st ja lõpeb dencOrigContent_count() tagastatud väärtusega.

- pEncData – EncruptedData objekt [nõutud].
- szOrigContentId – mingi väärtus loodava EncryptionProperty obejkti Id atribuudi jaoks [mitte nõutud]
- szName – puhver andmefaili nime jaoks

- szSize – puhver andmefaili suuruse jaoks
- szMime – puhver andmefaili maimi tüübi jaoks.
- szDfId – puhver andmefaili Id atribuudi väärtuse jaoks.
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencOrigContent_findByIndex(DEncEncryptedData* pEncData, int
origContIdx, char* szName, char* szSize, char* szMime, char* szDfId);
```

dencOrigContent_findByIndex()

Kontrollib kas krüpteeritud fail on digidoc dokument.

- pEncData – EncruptedData objekt [nõutud].
- Funktsioon tagastab 1 kui krüpteeritud fail on digidoc dokument.

```
EXP_OPTION int dencOrigContent_isDigiDocInside(DEncEncryptedData* pEncData);
```

dencOrigContent_registerDigiDoc()

Lisab krüpteeritud dokumendile sobiva maimi tüübi mis viitab sellele et sisuks on krüpteeritud digidoc dokument. Koostab iga digidoc objekti andmefaili (DataFile) objekti kohta EncryptionProperty objekti kuhu salvestatakse puhastekstina andmefaili nimi, suurus ja maimi tüüp.

- pEncData – EncryptedData objekt [nõutud].
- pSigDoc – SignedDoc objekt [nõutud].
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencOrigContent_registerDigiDoc(DEncEncryptedData* pEncData, SignedDoc*
pSigDoc);
```

dencMetaInfo_SetLibVersion()

Lisab krüpteeritud dokumendile EncryptionProperty objekti kuhu salvestatakse dokumendi koostanud teegi nimi ja versioon.

- pEncData – EncryptedData objekt [nõutud].
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencMetaInfo_SetLibVersion(DEncEncryptedData* pEncData);
```

dencMetaInfo_SetFormatVersion()

Lisab krüpteeritud dokumendile EncryptionProperty objekti kuhu salvestatakse dokumendi formaadi nimi ja versioon.

- pEncData – EncryptedData objekt [nõutud].
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencMetaInfo_SetFormatVersion(DEncEncryptedData* pEncData);
```

dencMetaInfo_GetLibVersion()

Loeb krüpteeritud dokumendi EncryptionProperty objektist dokumendi koostanud teegi nime ja versiooni.

- pEncData – EncryptedData objekt [nõutud].
- szLibrary – puhver teegi nime jaoks [nõutud].
- szVersion – puhver teegi versiooni jaoks [nõutud].
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencMetaInfo_GetLibVersion(DEncEncryptedData* pEncData, char*
szLibrary, char* szVersion);
```

dencMetaInfo_GetFormatVersion()

Loeb krüpteeritud dokumendi EncryptionProperty objektist dokumendi formaadi nime ja versiooni.

- pEncData – EncryptedData objekt [nõutud].
- szFormat – puhver formaadi nime jaoks [nõutud].
- szVersion – puhver formaadi versiooni jaoks [nõutud].
- Funktsioon tagastab veakoodi või ERR_OK.

```
EXP_OPTION int dencMetaInfo_GetFormatVersion(DEncEncryptedData* pEncData, char*
szFormat, char* szVersion);
```

MSSP Gateway functions

These functions provide an interface to MSSP Gateway webservice. MSSP Gateways is a SOAP webservice that provides a digital signature service on mobile phones.

ddocMsspConnect()

Initializes MSSP connection. Returns ERR_OK on success.

- pMssp - pointer to MSSP context structure [required]

```
EXP_OPTION int ddocMsspConnect(MSSP* pMssp);
```

ddocMsspDisconnect()

Cleanup MSSP connection. Returns ERR_OK on success.

- pMssp - pointer to MSSP context structure [required]

```
EXP_OPTION int ddocMsspDisconnect(MSSP* pMssp);
```

ddocMsspCleanup()

Cleanup MSSP connection but don't disconnect. Returns ERR_OK on success.

- pMssp - pointer to MSSP context structure [required]

```
EXP_OPTION int ddocMsspCleanup(MSSP* pMssp);
```

ddocMsspSignatureReq()

Sends an MSSP request to sign this data. Returns error code or SOAP_OK.

- pMssp - pointer to MSSP context structure [required]
- szPhoneNo - phone number on which to sign
- pHash - pointer to binary hash to sign
- nHashLen - length of hash data
- szDesc - description what user is signing (file name)

```
EXP_OPTION ddocMsspSignatureReq(MSSP* pMssp, const char* szPhoneNo,
              const char* pHash, int nHashLen, const char* szDesc);
```

ddocMsspStatusReq()

Sends an MSSP request to find the status of signature. Returns error code or SOAP_OK.

- pMssp - pointer to MSSP context structure [required]
- pMbufSignature - memory buffer to store signature value

```
EXP_OPTION int ddocMsspStatusReq(MSSP* pMssp, DigiDocMemBuf* pMbufSignature);
```

ddocMsspReadCertificate()

Reads the signers certificate based on the signers phone number. Returns error coder or ERR_OK

- szPhoneNo - phone number
- ppCert - address to store the certificate

```
EXP_OPTION int ddocMsspReadCertificate(const char* szPhoneNo, X509 **ppCert);
```

ddocGetSoapFaultString()

Rerieves SOAP fault string. Returns SOAP fault string or NULL

- pMssp - pointer to MSSP context structure [required]

```
EXP_OPTION int ddocGetSoapFaultString(MSSP* pMssp);
```

ddocGetSoapFaultCode()

Rerieves SOAP fault code. Returns SOAP fault code or NULL

- pMssp - pointer to MSSP context structure [required]

```
EXP_OPTION int ddocGetSoapFaultCode(MSSP* pMssp);
```

ddocGetSoapFaultDetail()

Rerieves SOAP fault detail. Returns SOAP fault detail or NULL

- pMssp - pointer to MSSP context structure [required]

```
EXP_OPTION int ddocGetSoapFaultDetail(MSSP* pMssp);
```

ddocConfMsspGetStatus()

Gets MSSP session status and returns status code. If you pass in a digidoc then the last signature will be finalized with signature value if available or removed in case of session error, timeout or users cancelling signature operation. Returns SOAP fault detail or NULL

- pMssp - pointer to MSSP context structure [required]
- pSigDoc - signed document object to be modified

```
EXP_OPTION int ddocConfMsspGetStatus(MSSP* pMssp, SignedDoc* pSigDoc);
```

ddocConfMsspSign()

Signs the document and gets return status back. Returns SOAP fault detail or NULL

- pMssp - pointer to MSSP context structure [required]
- pSigDoc - signed document object to be modified
- szPhoneNo - users phone number
- manifest - manifest or role
- city - signers address , city
- state - signers address , state or province
- zip - signers address , postal code
- country - signers address , country name
- szDigiDocFile - name of the file user signs

```
EXP_OPTION int ddocConfMsspSign(SignedDoc* pSigDoc, MSSP* pMssp, const char*
szPhoneNo, const char* manifest, const char* city, const char* state, const char* zip,
const char* country, const char* szDigiDocFile);
```

Veatöötlusfunktsioonid

Veatöötlusfunktsioonid võimaldavad hankida infot DigiDoc teegi funktsioonide töös tekkinud veasituatsioonide kohta. Veasituatsioone võidakse DigiDoci poolt registreerida kahel viisil:

- funktsiooni tagastusväärtusena. Sel juhul on võimalik DigiDocist küsida veakoodi tähendust tekstikujul.
- mälus hoitava veainfona. Veainfot saab kätte funktsiooni getErrorInfo abil. Mällu registreeritakse veainfo enamsti siis, kui funktsiooni tagastusväärtus ei kujuta endast veakoodi, vaid midagi muud, näiteks viita andmestruktuurile.

Mällu registreeritud vigade olemasolu saab kindlaks teha funktsiooni hasUnreadErrors abil või viimati väljakutsutud funktsiooni anomaalse tulemuse: nulline viit või veakood tagastusväärtusena.

Tagastusväärtusena antud veakoodi variandi puhul on võimalik, et registreeriti ka viimast veakoodi põhjustanud varemtoimunud vigu mällu. Seetõttu on mõistlik alati peale veakoodi avastamist ka mällu registreeritud vigu kontrollida.

getErrorString()

`Tagastab vea tekstilise kirjelduse vastavalt veakoodile ja keelele.`

- `code – veakood (funktsiooni poolt tagastatud või ErrorInfo sruktuurist)`

```
char* getErrorString(int code);
```

getErrorClass()

`Tagastab vea klassifikatsiooni.`

`Hetkel on registreeritud kolm vigade klassi:`

`NO_ERRORS – viga ei esinenud, puudub reageerimise vajadus.`

`TECHNICAL – mingi tehniline probleem.`

`USER - kasutaja poolt likvideertav probleem.`

`LIBRARY - teegisisene viga`

- code – veakood (funktsiooni poolt tagastatud või ErrorInfo sruktuurist)

```
ErrorClass getErrorClass(int code);
```

checkDigiDocErrors()

```
// kontrollib vigu ja trükib nad

// standardväljundisse. Tagastab viimase vea
```

```
int checkDigiDocErrors();
```

getErrorInfo()

```
Tagastab hiliseima, kuid veel lugemata vea ErrorInfo struktuuri.

Kui vigu ei ole registreeritud, siis tagastatakse 0.

ErrorInfo* getErrorInfo();
```

hasUnreadErrors()

```
Tagastab 1 kui eksisteerib veel lugemata vigu, 0 kui lugemata vigu
pole.

int hasUnreadErrors();
```

clearErrors()

```
Kustutab kõigi mälus olevate (nii loetud kui lugemata) vigade info.

v
```

DigiDoc teegis kasutatavad konstandid ja nende väärtused

Allkirjastamisega seotud konstandid

SIGNATURE_LEN 128

// kasutatud allkirja pikkus

DIGEST_LEN 20

// räsi pikkus

DIGEST_SHA1 0

// kasutusel olev räsi

CERT_DATA_LEN 2048

// sertifikaadi andmete suurim pikkus

X509_NAME_LEN 256

// sertifikaadi nimevälja (subject ja issuer) pikkus

SIGNATURE_RSA 0

//kasutusel olev allkiri

CONTENT_DETATCHED "DETATCHED"

CONTENT_EMBEDDED "EMBEDDED"

CONTENT_EMBEDDED_BASE64 "EMBEDDED_BASE64

// need konstandid näitavad kuidas andmefail on seotud //digidoci konteineriga

Vorminguga seotud konstandid

SK_PKCS7_1 "SK-PKCS#7-1.0"

SK_XML_1_NAME "SK-XML"

SK_XML_1_VER "1.0"

DIGIDOC_XML_1_1_VER "1.1"

DIGIDOC_XML_1_2_VER "1.2"

SK_NOT_VERSION "OCSP-1.0"

CHARSET_ISO_8859_1 "ISO-8859-1"

CHARSET_UTF_8 "UTF-8"

DIGEST_SHA1_NAME "sha1"

SIGN_RSA_NAME "RSA"

OCSP_NONCE_NAME "OCSP Nonce"

RESPID_NAME_VALUE "NAME"

RESPID_KEY_VALUE "KEY HASH"

OCSP_SIG_TYPE "sha1WithRSAEncryption"

FILE_FORMAT_ASN1 0

FILE_FORMAT_PEM 1

Veakoodid

ERROR_BUF_LENGTH 20

// Vigade puhvri suurus

ERR_OK 0

// See kood näitab , et vigu polnud

ERR_UNSUPPORTED_DIGEST 1

// Prooviti kasutada räsikoodi mida teek ei toeta, hetkel

// lubab teekkasutada ainult SHA1.

ERR_FILE_READ 2

// Ei saanud faili lugemiseks avada

ERR_FILE_WRITE 3

// Ei saanud faili kirjutamiseks avada

ERR_DIGEST_LEN 4

// Vale räsikoodi pikkus

ERR_BUF_LEN 5

// Liiga väike mälupuhvri pikkus

ERR_SIGNATURE_LEN 6

// Vale allkirja pikkus

ERR_PRIVKEY_READ 7

// Privaatvõtme lugemine ebaõnnestus

ERR_PUBKEY_READ 8

// Avaliku võtme lugemine ebaõnnestus

ERR_CERT_READ 9

// Sertifikaadi lugemine ebaõnnestus

ERR_SIGNEDINFO_CREATE 10

// Ei suutnud tekitada allkirja objekti

ERR_SIGNEDINFO_DATA 11

// Ei suutnud tekitada allkirja objekti

ERR_SIGNEDINFO_FINAL 12

// Ei suutnud tekitada allkirja objekti

ERR_UNSUPPORTED_FORMAT 13

// Vale allkirjastatud dokumendi formaat

ERR_BAD_INDEX 14

// Vale indeks

ERR_TIMESTAMP_DECODE 15

// Ajatempli dekodeerimine ebaõnnestus

ERR_DIGIDOC_PARSE 16

// Viga dokumendi süntaksianalüüsil

ERR_UNSUPPORTED_SIGNATURE 17

// Vale allkirja tüüp

ERR_CERT_STORE_READ 18

// Ei suutnud lugeda sertifikaati sertifikaatide

// hoidlast

ERR_SIGPROP_DIGEST 19

// Vale allkirja omaduste räsikood

ERR_COMPARE 20

// Vale allkiri

ERR_DOC_DIGEST 21

// Vale dokumendi räsikood

ERR_MIME_DIGEST 22

// Vale dokumendi tüübi räsikood

ERR_SIGNATURE 23

// Vale allkiri

ERR_CERT_INVALID 24

// Sobimatu sertifikaat

ERR_OCSP_UNSUCCESSFUL 25

// OCSP päring ebaõnnestus

ERR_OCSP_UNKNOWN_TYPE 26

// Tundmatu OCSP tüüp

ERR_OCSP_NO_BASIC_RESP 27

// OCSP_BASIC_RESP puudub

ERR_OCSP_WRONG_VERSION 28

// Vale OCSP versioon

ERR_OCSP_WRONG_RESPID 29

// OCSP vastuse ID on vale

ERR_OCSP_ONE_RESPONSE 30

// OCSP vastuste arv ei klapi

ERR_OCSP_RESP_STATUS 31

// Vale OCSP vastuse staatus

ERR_OCSP_NO_SINGLE_EXT 32

// Ebakorrektne OCSP laiendus

ERR_OCSP_NO_NONCE 33

// OCSP vastuses puudub NONCE

ERR_NOTARY_NO_SIGNATURE 34

// Puudub allkiri mille kohta OCSP päringut

// teostada

ERR_NOTARY_SIG_MATCH 35

// Notari allkiri vigane

ERR_WRONG_CERT 37

// Sobimatu sertifikaat

ERR_NULL_POINTER 38

// Nulline viit

ERR_NULL_CERT_POINTER 39

// Nulline sertifikaadi viit

ERR_NULL_SER_NUM_POINTER 40

// Nulline sertifikaadi numbri viit

ERR_NULL_KEY_POINTER 41

// Nulline võtmeviit

ERR_EMPTY_STRING 42

// Tühi string

ERR_BAD_DATAFILE_INDEX 43

// Andmefaili indeks on piiridest väljas

ERR_BAD_DATAFILE_COUNT 44

// Andmefailide loendur on vale

ERR_BAD_ATTR_COUNT 45

// Atribuutide loendur on vale

ERR_BAD_ATTR_INDEX 46

// Atribuudi indeks on piiridest väljas

ERR_BAD_SIG_INDEX 47

// Allkirja indeks on piiridest väljas

ERR_BAD_SIG_COUNT 48

// Allkirjade loendur on vale

ERR_BAD_ROLE_INDEX 49

// Rolli indeks on piiridest väljas

ERR_BAD_DOCINFO_COUNT 50

// Dok. info loendur on vale

ERR_BAD_DOCINFO_INDEX 51

// Dok. info indeks on piiridest väljas

ERR_BAD_NOTARY_INDEX 52

// Notari indeks on piiridest väljas

ERR_BAD_NOTARY_ID 53

// Vale notari ID

ERR_BAD_NOTARY_COUNT 54

// Notarite loendur on vale

ERR_X509_DIGEST 55

// X509 räsikoodi arvutus ebaõnnestus

ERR_CERT_LENGTH 56

// Sertifikaadi pikkus on vale

ERR_PKCS_LIB_LOAD 57

// PKCS #11 DLL-i laadimine ebaõnnestus

ERR_PKCS_SLOT_LIST 58

// Ebaõnnestus PKCS #11 slottide küsimine

ERR_PKCS_WRONG_SLOT 59

// Sellist PKCS #11 slotti ei ole olemas

ERR_PKCS_LOGIN 60

// Kaart ei ole sisestatud, PIN on vale või

// blokeeritud

ERR_PKCS_PK 61

// Ei suuda leida EstID salajase võtme asukohta

ERR_PKCS_CERT_LOC 62

// Ei suuda lugeda EstID allkirjastamise sertifikaati

ERR_PKCS_CERT_DECODE 63

// Sertifikaadi dekoreerimine ebaõnnestus

ERR_PKCS_SIGN_DATA 64

// Allkirjastamine EstID kaardiga ebaõnnestus

ERR_PKCS_CARD_READ 65

// EstID kaardi lugemine ebaõnnestus

ERR_CSP_NO_CARD_DATA 66

// EstID kaart ei ole kättesaadav

ERR_CSP_OPEN_STORE 67

// Ei õnnestu avada süsteemi sertifikaatide hoidlat

ERR_CSP_CERT_FOUND 68

// Ei leitud sertifikaati, kontrollige kas sertifikaat

// on registreeritud

ERR_CSP_SIGN 69

// Allkirjastamine CSP-ga ebaõnnestus

ERR_CSP_NO_HASH_START 70

// Ei suuda alustada CSP räsi arvutamist

ERR_CSP_NO_HASH 71

// CSP räsi arvutamine ebaõnnestus

ERR_CSP_NO_HASH_RESULT 72

// Ei suuda lugeda CSP räsi tulemust

ERR_CSP_OPEN_KEY 73

// CSP ei suuda avada kaardi võtit

ERR_CSP_READ_KEY 74

// CSP ei suuda lugeda kaardi võtit

ERR_OCSP_SIGN_NOT_SUPPORTED 75

// Valitud OCSP allkirjastamise viis ei toetata

ERR_OCSP_SIGN_CSP_NAME 76

// Ei suuda lisada allakirjutaja nime OCSP päringule

ERR_CSP_CERT_DECODE 77

// Sertifikaadi dekoreerimine ebaõnnestus

ERR_OCSP_SIGN_PKCS_NAME 78

// Ei suuda lisada allkirjutaja nime OCSP päringule

ERR_OCSP_SIGN_OSLL_CERT 79

// Ei suuda lisada sertifikaati OCSP päringusse

ERR_OCSP_SIGN 80

// Ei suuda allkirjastada OCSP päringut

ERR_CERT_ISSUER 81

// Tundmatu autoriteedi poolt välja antud

// sertifikaat, või vale allkiri sertifikaadil

ERR_OCSP_PKCS12_CONTAINER 82

// Ei suuda avada PKCS#12 konteinerit

ERR_MODIFY_SIGNED_DOC 83

// Ei saa muuta allkirjastatud faili. Eemaldage

// enne allkirjad!

ERR_NOTARY_EXISTS 84

// Ei saa kustutada allkirja kui kehtivuskinnitus

//on olemas. Eemaldage esmalt vastav

// kehtivuskinnitus!

ERR_UNSUPPORTED_CERT_SEARCH 85

// Tundmatu otsigu meetod

ERR_INCORRECT_CERT_SEARCH 86

// Vigane otsingu muster

ERR_BAD_OCSP_RESPONSE_DIGEST 87

// Kehtivuskinnituse kontrollkood on vale

ERR_LAST_ESTID_CACHED 88

// Vale sertifikaat puhvris, proovige uuesti.

ERR_BAD_DATAFILE_XML 89

// Andmed ei tohi sisaldada XML faili esimest rida

ERR_UNSUPPORTED_VERSION 90

// Dokument on loodud uuema tarkvara

// versiooniga. Palun uuendage tarkvara!

ERR_UNSUPPORTED_CHARSET 91

// mitte toetatud kooditabel

ERR_PKCS12_EXPIRED 92 // Juurdepääsutõendi kehtivusaeg on lõppenud!

ERR_CSP_USER_CANCEL 93

// Kasuja loobus sertifikaadi valikust

ERR_CSP_NODEFKEY_CONTAINER 94

// Ei leitud vaikevõtme konteinerit

ERR_CONNECTION_FAILURE 95

// Ühendus ebaõnnestus

ERR_WRONG_URL_OR_PROXY 96

//Vale URL või proksi

ERR_NULL_PARAM 97

// Kohustuslik parameeter oli NULL

ERR_BAD_ALLOC 98

// Viga mäluhõlvamisel

ERR_CONF_FILE 99

// Viga konfiguratsioonifaili avamisel

ERR_CONF_LINE

// Viga konfiguratsioonifailis

ERR_MAX 102

// viimane kood omab ainult teegi sisest tähendust

Kasutusel olevad keeled

DDOC_LANG_ENGLISH 0

DDOC_LANG_ESTONIAN 1

DDOC_NUM_LANGUAGES 2

OCSP päringu allkirjastamise tüübi identifikaatorid

OCSP_REQUEST_SIGN_PKCS12 5

// OCSP päring allkirjastatakse eeldusel, et privaatne võti

// ja sertifikaat on PKCS #12 tüüpi konteineris.

Sertifikaatide otsingu kohad

Nende konstatntide kasutamise kohta saab infot näitest "Kuidas otsida sertifikaate"

CERT_SEARCH_BY_STORE 1

// sertifikaat loetakse MS Certificate Storest ehk Windowsi

// sertifikaatide hoidlast

CERT_SEARCH_BY_X509 2

// sertifikaat loetakse PEM failist

CERT_SEARCH_BY_PKCS12 3

// sertifikaat loetakse PKCS#12 tüüpi konteinerist.

Sertifikaatide otsingu kriteeriumid

Nende konstantide abil saab juhtida sertifikaatide otsimist Windowsi sertifikaatide hoidlast.

CERT_STORE_SEARCH_BY_SERIAL 0x01

CERT_STORE_SEARCH_BY_SUBJECT_DN 0x02

CERT_STORE_SEARCH_BY_ISSUER_DN 0x04

CERT_STORE_SEARCH_BY_KEY_INFO 0x08

Toetatud kaartide nimed

Kasutatakse hetkel vaid teegi siseselt.

EST_EID_CSP "EstEID Card CSP"

EST_AEID_CSP "Gemplus GemSAFE Card CSP"

EST_AEID_CSP_WIN "Gemplus GemSAFE Card CSP v1.0"