

DigiDoc C-teegi kirjeldus koos kasutusjuhistega

Dokumendi versioon: 1.89, 14.06.2004

Sisukord:

Viited ja lühendid.....	7
DigiDoc C-teegi kasutamine.....	8
C-teegi ülevaade	8
Sissejuhatus.....	8
Teegi failid.....	8
Teegi komponendid.....	9
Erinevused teegi versioonide 1.89 ja 1.88 vahel.....	10
Erinevused teegi versioonide 1.88 ja 1.80 vahel.....	10
Erinevused teegi versioonide 1.80 ja 1.77 vahel.....	11
Erinevused teegi versioonide 1.79 ja 1.77 vahel.....	11
Erinevused teegi versioonide 1.77 ja 1.76 vahel.....	12
Erinevused teegi versioonide 1.76 ja 1.66 vahel.....	12
Erinevused teegi versioonide 1.66 ja 1.52 vahel.....	13
Erinevused teegi versioonide 1.52 ja 1.41 vahel.....	14
Erinevused teegi versioonide 1.52 ja 1.59 vahel.....	14
Juhised enam esinevate probleemide kiireks lahendamiseks.....	15
Kuidas luua uusi DigiDoc'e.....	15
Kuidas lisada kehtivuskinnitust ?.....	18
Kuidas otsida sertifikaate DigiDoc'i funktsioonides ?.....	19
Kuidas avada ja lugeda DigiDoc'e.....	20
Kuidas lisada allkirja DigiDoc'ile.....	21
Näiteprogrammid DigiDoc'i C-teegi kasutamiseks.....	23
CreateDoc.c.....	23
ListSignatures.c.....	24
DigiDoc teek.....	26
Andmestruktuurid.....	26
SignedDoc.....	26
DataFile.....	26
DocInfo.....	27
SignatureProductionPlace.....	27
SignerRole.....	27
SignatureInfo.....	28
NotaryInfo.....	28
Timestamp.....	29
PolicyIdentifier.....	29
CertSearch.....	29
CertSearchStore.....	30
ErrorInfo.....	30
CSProvider.....	30
CertItem.....	30
Üldised- ja administreerimisfunktsioonid.....	31
getLibName().....	31
getLibVersion().....	31
getSupportedFormats().....	31

getSupportedFormatsAndVersions()	31
initDigiDocLib()	31
finalizeDigiDocLib()	31
trim()	31
ascii2utf8()	31
utf82ascii()	32
unicode2ascii()	32
convertStringToTimestamp()	32
convertTimestampToString()	32
Timestamp_new()	32
Timestamp_free()	33
Üldised krüptograafiafunktsioonid	33
bin2hex()	33
calculateFileDigest()	33
calculateFileSignature()	33
verifyFileSignature()	34
signData()	34
calculateDigest()	34
encode()	34
decode()	35
Üldised XMLi genereerivad funktsioonid	35
createTimestamp()	35
createXMLSignedInfo()	35
Allkirjastatud dokumendi halduse funktsioonid	35
getSimpleFileName()	35
SignedDoc_new()	36
SignedDoc_free()	36
getCountOfDataFiles()	36
getDataFile()	36
getDataFileWithId()	36
DataFile_new()	36
DataFile_free()	37
getCountOfDataFileAttributes()	37
addDataFileAttribute()	37
getDataFileAttribute()	37
calculateDataFileSizeAndDigest()	37
getCountOfSignatures()	38
getSignature()	38
getSignatureWithId()	38
SignatureInfo_new()	38
setSignatureProductionPlace()	38
addSignerRole()	38
getCountOfSignerRoles()	39
getSignerRole()	39
SignatureInfo_delete()	39
SignatureInfo_free()	39
addDocInfo()	39

DocInfo_free()	40
getCountOfDocInfos()	40
getDocInfo()	40
getDocInfoWithId()	40
setDocInfoDigest()	40
setDocInfoMimeDigest()	40
addAllDocInfos()	40
calculateSigInfoSignature()	41
calculateSignatureWithEstID()	41
calculateSigInfoSignatureWithCSPEstID()	41
GetSignParametersWithEstIdCSP()	41
getCountOfNotaryInfos()	42
getNotaryInfo()	42
getNotaryWithId()	42
getNotaryWithSigId()	42
NotaryInfo_new()	42
NotaryInfo_new_file()	43
NotaryInfo_free()	43
NotaryInfo_delete()	43
createXMLSignedProperties()	43
calculateSignedPropertiesDigest()	43
calculateSignedInfoDigest()	43
setSignatureCertFile()	44
setSignatureCert()	44
setSignatureValueFromFile()	44
setSignatureValue()	44
Allkirjastatud dokumendi lugemine ja kirjutamine	44
createSignedDoc()	44
readSignedDoc()	45
extractDataFile()	45
MS sertifikaadihoidla funktsioonid	45
CSPProvider_new ()	45
CSPProvider_free()	45
isActiveCSPProvider()	45
getCurrentCSPProvider ()	45
setCurrentCSPProvider ()	46
findIssuerCertificatefromMsStore()	46
resolveCertChain_ByCertStore ()	46
Sertifikaatide funktsioonid	46
getSignCertData()	46
findCertificate()	46
getNotCertData()	46
getCertIssuerName()	46
getCertSubjectName()	47
getCertSerialNumber()	47
GetCertSerialNumber()	47
getCertNotBefore()	47

getCertNotAfter().....	47
saveCert().....	47
decodeCert().....	47
encodeCert().....	48
CertSearchStore_new().....	48
CertSearchStore_free().....	48
CertSearch_new().....	48
CertSearch_free().....	48
CertList_free ().....	48
readCertPolicies().....	48
PolicyIdentifiers_free().....	49
isCompanyCPSPolicy().....	49
Allkirja kontrolli funktsioonid.....	49
compareByteArrays().....	49
verifySigDocDigest().....	49
verifySigDocMimeDigest().....	49
verifySigDocSigPropDigest().....	50
verifySignatureInfo().....	50
verifySignatureInfoCERT().....	50
verifySignatureInfo_ByCertStore().....	51
verifySigDoc().....	51
verifySigDocCERT().....	51
verifySigDoc_ByCertStore ().....	52
isCertValid().....	52
isCertSignedBy().....	52
isCertSignedByCERT().....	52
verifySigCert().....	52
verifyNotaryInfo().....	53
verifyNotaryInfoCERT().....	53
verifyNotaryInfo_ByCertStore().....	53
verifyNotCert ().....	53
verifyNotaryDigest().....	53
writeOCSPRequest().....	54
getConfirmation().....	54
calculateNotaryInfoDigest().....	54
getSignerCode().....	54
Veatöötlusfunktsioonid.....	55
getErrorString().....	55
getErrorClass().....	55
checkDigiDocErrors().....	56
getErrorInfo().....	56
hasUnreadErrors().....	56
clearErrors().....	56
DigiDoc teegis kasutatavad konstandid ja nende väärtused.....	56
Allkirjastamisega seotud konstandid.....	56
Vorminguga seotud konstandid.....	57
Veakoodid.....	57

Kasutusel olevad keeled	62
OCSF päringu allkirjastamise tüübi identifikaatorid.....	62
Sertifikaatide otsingu kohad.....	62
Sertifikaatide otsingu kriteeriumid.....	63
Toetatud kaartide nimed.....	63

Käesolev dokument kirjeldab DigiDoc'i C-teeki , mida kasutab saavad kasutada kõik DigiDoc süsteemi rakendused. DigiDoc dokument on esitatud XML kujul ning põhineb rahvusvahelistel standarditel XML-DSIG ja ETSI TS 101 903. DigiDoc'i teegi COM objektide versioon on dokumenteeritud eraldi.

Viited ja lühendid

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	Eesti Digitaallkirja Seadus
ESTEID	Eesti ID kaart
CSP (ka MS CSP)	Microsoft Crypto Service Provider
PKCS#11	RSA Laboratories Cryptographic Token Interface Standard http://www.rsasecurity.com/rsalabs/PKCS/index.html

DigiDoc C-teegi kasutamine

C-teegi ülevaade

Sissejuhatus

DigiDoc C-teek sobib kasutamiseks DigiDoc'i vormingut ja ESTEID kasutavates rakendustes. Teegis on funktsioonid DigiDoc formaadis dokumentide loomiseks, kustutamiseks, haldamiseks ja allkirjastamiseks. Teegi kasutamise hlbustamiseks on antud dokumendi lppu lisatud mned näited sagedamate probleemide kiireks lahendamiseks. Ta sõltub järgmistest teekidest:

- **OpenSSL** – version 0.9.7 või uuem. Seda saab siit: <ftp://ftp.openssl.org/snapshot/>
- **libxml2** – versioon 2.5.1 või uuem. Seda saab siit: <ftp://ftp.gnome.org/pub/GNOME/stable/sources/libxml/>, <http://www.xmlsoft.org/> ja valmis binary failid siit: <http://www.zlatkovic.com/projects/libxml/index.html>

Kui soovite teeki kasutada ka ESTEID kaardiga allkirjastamiseks on tarvilik kaardilugeja koos tarvilike ohjurprogrammidega. Teek toeatab kiipkaardi kasutamist läbi PKCS#11 standardi (platvormil Win32) ja läbi Microsofti CSP.

DigiDoc teek üritab luua vahendid soovitud formaadis failide lugemisele ja kirjutamisele, sidumata kasutajat seejuures mingi kindla küptograafiapaketi, faili füüsilise formaadi (XML vs. PKCS#7) või muu teegi versiooniga. Seepärast kasutatakse andmete salvestamiseks programeerimiskeele C struct -e, mis ei ole sõltuvad OpenSSL või XML teegi kasutatavatest andmestruktuuridest.

Teegi failid

Teeki on kompileeritud Fedora Core 1 ja Mandrake 9.2 peal ning WinNT 4.0 ja Win2000 peal. M nede teegi funktsioonide jaoks on tarvilik kaardilugeja ning ESTEID kiipkaart

DigiDoc C-teegi koosseisus kuuluvad neli algkoodi faili:

1. **DigiDocLib.c** - peamine algkoodi fail kuhu on koondatud peaaegu kõik funktsioonid, seal hulgas kõik DigiDoc'i kontaineri ja selle elementidega tegelevad funktsioonid, samuti kontrolli ja kehtivuskinnituse funktsioonid. Mõned eelkõige allkirjastamisega tegelevad funktsioonid kasutavad ESTEID kaardi võimalusi ning on seega kaardilugeja ohjur programmide kaudu platvormist sõltuvad, realiseeritud on PKCS#11 tüüpi kaardilugeja Windows platvormil ja MS CSP (samuti Windows platvormil).

Platvormist sõltuvad avalikud funktsioonid on:

- **GetSignParametersWithEstIdCSP()** - kasutatakse mitmesuguste

ESTEID parameetrite küsimiseks läbi CSP.

- `calculateSignatureWithEstID()` - lisab digidoc'ile ESTEID allkirja läbi PKCS#11 tüüpi kaardilugeja.
- `calculateSigInfoSignatureWithCSPEstID()` - lisab digidoc'ile ESTEID allkirja läbi CSP.

Need funktsioonid sisaldavad kutseid `EstIDLib.c`'s asuvatele platvormist ja ESTEID'st sõltuvatele funktsioonidele, ning muutuksid mõttetuteks kui `EstIDLib.c`'s asuvad funktsioonid poleks neile kätte saadavad. Peale nende on ka mõned funktsioonid mida ainult teegi siseselt kasutatakse ja mõnede funktsioonide sees on osa blokke mis mitte windowsi platvormidel kompileerimisest välja jäävad.

Teisel platvormidel kompileerides tuleks vältida eelprotsessori konstante `WIN32` ja `WIN32_CSP` mis vastava sõltuvuse koodi sisse kompileerivad.

2. `DigiDocLib.h` - Selles failis on deklareeritud kõik `DigiDoc`'i avalikud funktsioonid ja konstandid
3. `DigiDocPKCS11.c` - Fail sisaldab funktsioone ESTEID kaardiga vajalike operatsioonide teostamiseks Windows'i ja Linux platvormil läbi PKCS#11 ühilduvate kaardilugejate.
4. `DigiDocPKCS11.h` - Fail sisaldab `DigiDocPKCS11.c`'s defineeritud funktsioonide deklaratsioone.
5. `DigiDocConfig.c` - sisaldab konfiguratsioonifaili lugemiseks vajalikke funktsioone ja lihtsustatud allkirjastamis ning allkirja ja kehtivuskinnituse kontrollfunktsioone.
6. `DigiDocConfig.h` - sisaldab `DigiDocConfig.h`-s deklareeritud funktsioone
7. `DigiDocError.c` – sisaldab veatöötlusfunktsioone ja vigade kirjeldusi.
8. `DigiDocError.h` – sisaldab veatöötlusfunktsioonide ja veakoodide deklareeratsioone.
9. `digidoc.c` - käsureautiliit digidoc failidega seotud operatsioonideks ja samas ka näiteprogramm.

Teegi komponendid

Teek koosneb kolme liiki komponentidest:

- Andmestruktuurid - need kasutavad C keele struktuure andmete paremaks haldamiseks, enamus neist peegeldab vastavaid `DigiDoc`'i tag'e aga on ka mõned teegi tööd hõlbustavad struktuurid millel puudub vaste `DigiDoc`'is, näiteks `ErrorInfo` või `CertSearch` jt. Andmestruktuurid on kirjeldatud samanimelises peatükis allpool ja deklareeritud on nad algfailis `DigiDocLib.h`.
- Konstandid - Oma töös kasutab teek palju konstante, sealhulgas veakoode. Konstandid on sammuti defineeritud failides `DigiDocLib.h` ja `DigiDocError.h` , neile on pühendatud oma peatükk käesolevas dokumendis.

- Funktsioonid - need on defineeritud teegi *.c'i failides. Avalikku huvi pakkuvad funktsioonid on ka deklareeritud DigiDocLib.h failis. Need viimased võib jaotada:
 - **Üldised- ja administreerimisfunktsioonid** - siia all kuuluvad funktsioonid teegi enda kohta, samuti mõned üldised teisendus funktsioonid.
 - **Üldised krüptograafiafunktsioonid** - siia hulka kuuluvad mõned räsikoodi arvutavad funktsioonid ja üldised allkirjastamise funktsioonid, samuti mõned OpenSSL kasutavad teisendus funktsioonid.
 - **Üldised XMLi genereerivad funktsioonid** - siin all on funktsioonid mis sobisid määratluse XML'i genereerivad kuid "vähe" DigiDoc'i spetsiifilised funktsioonid.
 - **Allkirjastatud dokumendi halduse funktsioonid** - see on suurim funktsioonide klass kuhu kuuluvad uute struktuuride lisamise ja kasutamise funktsioonid aga ka struktuuridelt info küsimise ja ESTEID'ga allkirjastamise funktsioonid.
 - **Allkirjastatud dokumendi lugemine ja kirjutamine** - siin on paar üldist funktsiooni mis tegelevad DigiDoc'i kirjutamise ja lugemisega.
 - **Sertifikaatide funktsioonid** - siia klassi kuuluvad sertifikaatide lugemise, otsimise ja nendelt info küsimise funktsioonid.
 - **Allkirja kontrolli funktsioonid** - selles klassis asub enamus verifiy* funktsioonidest mis tegelevad DigiDoci'i struktuuri erinevate elementide kontrollimisega, samuti mõned üldised kontrolli funktsioonide poolt kasutatavad funktsioonid.
 - **Veatöötlusfunktsioonid** - selles viimases kuid tähtsas rühmas on veatöötlusega seotud funktsioonid.

Erinevused teegi versioonide 1.89 ja 1.88 vahel

- Parandasin funktsioone handleStartDataFile() ja generateDataFileXML() lubamaks sümboli '&' kasutamiste andmefailide nimedes.

Erinevused teegi versioonide 1.88 ja 1.80 vahel

- Veakoodid ERR_CERT_STORE_READ ja ERR_CERT_READ muudeti USER kategooria vigadeks, sest kui sertifikaati ei leia sertstorest või failist siis on midagi konfiguratsiooniga valesti.
- Parandatud bugid funktsioonis convertStringToTimeT()
- Lisatud veakood ERR_OCSP_WRONG_URL kui responderi URL on küll sisestatud aga ei osuta OCSP responderile. See on samuti USER klassi viga.
- Parandatud bugi funktsioonis setSignatureValueFromFile()
- Parandatud bugi funktsioonis getSignerCode()
- Struktuuride SignatureInfo ja NotaryInfo liige "long nCertSerial" muutus "char* szCertSerial" sest osadel sertifikaatidel oli nii suur number mis long andmetüüpi ära ei mahtunud.

Erinevused teegi versioonide 1.80 ja 1.77 vahel

- Dokumendid formaadis DIGIDOC-XML ja versioonis 1.3 peavad nüüd kasutama SignedDoc namespace. Muudetud funktsioonide generateDataFileXML(), createSignedXMLDoc() ja handleStartDataFile() sisu aga mitte liidest.
- Veaparandused funktsioonides unicode2ascii() ja getSignerLastName().

Erinevused teegi versioonide 1.79 ja 1.77 vahel

- Veatöötlusfunktsioonid ja konstandid viidud üle failidesse DigiDocError.c ja DigiDocError.h
- Veajadade säilitamine/töötlemine tehtud lõimepõhiseks ja kasutatavaks erinevates lõimedes samaaegselt.
- Lisatud funktsioon getErrorClass ja veaklassid NO_ERRORS, TECHNICAL, USER ja LIBRARY, veaklassid seostatud veakoodidega.
- Lisatud veakoodid ERR_NULL_PARAM, ERR_BAD_ALLOC, ERR_CONF_FILE, ERR_CONF_LINE.
- Kogu teegi ulatuses rakendatud veasituatsioonide registreerimine.
- Objekte tekitavad funktsioonid tagastavad objektiivida asemel veakoodi.
- Muudetud liides: Timestamp *Timestamp_new() -> int Timestamp_new()
- Muudetud liides: SignedDoc *SignedDoc_new() -> int SignedDoc_new()
- Muudetud liides: DataFile *DataFile_new() -> int DataFile_new()
- Muudetud liides: SignatureInfo *SignatureInfo_new() -> int SignatureInfo_new()
- Muudetud liides: NotaryInfo *NotaryInfo_new() -> int NotaryInfo_new()
- Muudetud liides: NotaryInfo *NotaryInfo_new_file() -> int NotaryInfo_new_file()
- Muudetud liides: X509* ReadCertificate() -> int ReadCertificate()
- Muudetud liides: long GetCertSerialNumber() -> int GetCertSerialNumber()
- Muudetud liides: long ReadCertSerialNumber() -> int ReadCertSerialNumber()
- Muudetud liides: EVP_PKEY* ReadPublicKey() -> int ReadPublicKey()
- Muudetud liides: EVP_PKEY* GetPublicKey() -> int GetPublicKey()
- Muudetud liides: EVP_PKEY* ReadPrivateKey() -> int ReadPrivateKey()
- Muudetud liides: RSA* ReadRSAPrivateKey() -> int ReadRSAPrivateKey()
- Muudetud liides: void convertStringToTimestamp() -> int convertStringToTimestamp()
- Muudetud liides: void convertTimestampToString() -> int convertTimestampToString()
- Muudetud liides: ASN1_GENERALIZEDTIME* str2asn1time() -> int str2asn1time()
- Muudetud liides: void setString() -> int setString()
- Muudetud funktsiooni nimi: int countDataFiles() -> int getCountOfDataFiles()
- Muudetud funktsiooni nimi: int countDataFileAttributes() -> int getCountOfDataFileAttributes()
- Muudetud liides: void addDataFileAttribute() -> int addDataFileAttribute()
- Muudetud liides: void getDataFileAttribute() -> int getDataFileAttribute()
- Muudetud funktsiooni nimi: int countSignatures() -> int getCountOfSignatures()
- Muudetud liides: void setSignatureProductionPlace() -> int setSignatureProductionPlace()

- Muudetud liides: void addSignerRole() -> int addSignerRole()
- Muudetud funktsiooni nimi: int countSignerRoles() -> int getCountOfSignerRoles()
- Muudetud liides: DocInfo* addDocInfo() -> int addDocInfo()
- Muudetud funktsiooni nimi: int countDocInfos() -> int getCountOfDocInfos()
- Muudetud liides: void addAllDocInfos() -> int addAllDocInfos()
- Muudetud liides: X509* decodeCertificateData() -> int decodeCertificateData()
- Muudetud liides: X509* decodeCertificatePEMData() -> int decodeCertificatePEMData()
- Muudetud liides: OCSP_RESPONSE* decodeOcspResponsePEMData() -> int decodeOcspResponsePEMData()
- Muudetud liides: OCSP_RESPONSE* ReadOCSPResponse() -> int ReadOCSPResponse()
- Muudetud liides: void WriteOCSPResponse() -> int WriteOCSPResponse()
- Muudetud liides: OCSP_REQUEST* ReadOCSPRequest() -> int ReadOCSPRequest()
- Muudetud liides: void WriteOCSPRequest() -> int WriteOCSPRequest()
- Muudetud funktsiooni nimi: int countNotaryInfos() -> int getCountOfNotaryInfos()
- Muudetud liides: void writeCertToXMLFile() -> int writeCertToXMLFile()
- Muudetud liides: void writeOcspToXMLFile() -> int writeOcspToXMLFile()
- Muudetud liides: void extractDataFile() -> int extractDataFile()
- Eemaldatud copyDataFile()
- Muudetud liides: void* getSignCertData() -> X509* getSignCertData()
- Muudetud liides: void* getNotCertData() -> X509* getNotCertData()
- Muudetud liides: X509_STORE *setup_verifyCERT() -> int setup_verifyCERT()
-

Erinevused teegi versioonide 1.77 ja 1.76 vahel

- Teek hoiab nüüd kõiki andmeid mälus UTF-8 kodeeringus. Faili kirjutades ja sealt lugedes enam muudatusi ei tehta. Kasutaja peab andmed edastama ja saadud andmeid konverteerima vastavalt vajadusele.
- Eemaldatud parameeter szCharset funktsioonilt addDataFileAttribute()
- Eemaldatud parameeter szCharset funktsioonilt setSignatureProductionPlace()
- Lisatud parameeter szFileNameCharset funktsioonile DataFile_new() ja extractDataFile()

Erinevused teegi versioonide 1.76 ja 1.66 vahel

- Teek toetab nüüd ka formaadi versiooni "1.3" - DIGIDOC-XML 1.3 versioon on loodud eesmärgiga parandada süntaktilisi vigu, et tagada parem ühilduvus XadES spetsifikatsiooniga. Muudetakse järgmisi elemente:
 - <QualifyingProperties> - versioon 1.2 ei oma mingeid atribuute. Tegelikult peab omama atribuute xmlns="http://uri.etsi.org/01903/v1.1.1#" ja Target="#<allkirja-id>"

- <SignedProperties> - versioon 1.2 omab atribuute
xmlns="http://www.w3.org/2000/09/xmldsig#" Id="S0-SignedProperties"
Target="#S0". Tegelikult on lubatud ainult Id-atribuut.
- <IssuerSerial> - sisaldab versioonis 1.2 allkirjastaja serdi väljaandja poolt omistatud seerianumbrit. Tegelikult peaks sisaldama alamelemente kujul:

```
<IssuerSerial>
  <X509IssuerName xmlns="http://www.w3.org/2000/09/xmldsig#">
    <!-- allkirjastaja sertifikaadi issuer DN -->
  </X509IssuerName>
  <X509SerialNumber>
    <!-- allkirjastaja sertifikaadi seerianumber -->
  </X509SerialNumber>
</IssuerSerial>
```
- dateTime andmetüübi viga: kuupäeva osas peab kasutama '.' asemel '-'. Ehk siis näiteks: 2003-11-06T14:23:49Z, mitte 2003.11.06T14:23:49Z. Kehtib elementide <SigningTime> ja <ProducedAt> kohta.
- Elemendis <Cert> pole parameeter „Id“ lubatud
- Elemendis <UnsignedProperties> pole parameeter „Target“ lubatud
- Elementide <CompleteCertificateRefs> ja <Cert> vahel peab täiendavalt olema element <CertRefs>.
- Elementide <RevocationValues> ja <EncapsulatedOCSPvalue> vahel peab täiendavalt olema element <OCSPValues>.
-

Erinevused teegi versioonide 1.66 ja 1.52 vahel

- setup_verifyCERT() - votab vastu suvalise pikkusega sertide array. Ennem tapselt 2 CA-d
- setup_verify() - pole enam vaja, selle asemel setup_verifyCERT()
- verifyNotaryInfoCERT() - votab vastu suvalise pikkusega sertide array
- verifyNotaryInfo() - loeb serdid failist ja siis kasutab verifyNotaryInfoCERT(). Ennem duplitseeritud kood.
- getConfirmation() - votab vastu suvalise pikkusega sertide array.
- isCertValid() - lisandus parameeter "time_t tDate". Timestamp millal serdi kehtivust kontrollida. Nyyd kasutatakse muudest funktsioonidest et kontrollida serdi kehtivust allkirjastamise ajal ja mitte praegusel hetkel. Parandab aastaarvu kruttimise vea.
- verifySignatureInfo() - kasutab uut isCertValid() allkirjastamise kuupäevaga
- signOCSPRequestPKCS12() - kontrollib PKCS#12 konteineri serdi aegumist hetkel kui OCSP-d saadetakse
- verifySigDocCERT() - votab vastu suvalise pikkusega sertide array.
- verifySignatureInfoCERT() - kasutab uut isCertValid() allkirjastamise kuupäevaga
- COM jaoks kasutatakse isCertValid() puhul hetkelist kuupäeva. Siia pole parameetrit tDate lisatud. COM tegelikult seda ise valjakutsuma ei pea vaid seda funktsiooni kasutatakse teegi seest allkirja kontrollimisel.
- decodeHex() - lisandunud

- `get_authority_key()` - lisandunud
- `createOCSPCertid()` - lisandunud
- `createOCSPRequest()` - Enam ei vaja responderi CA-d et OCSP requesti teha

Erinevused teegi versioonide 1.52 ja 1.41 vahel

- Lubatud on nüüd ka formaadi versioon "1.2"
- Teek hoiab mälus andmeid tähestikus "ISO-8859-1". Võimalik on lisada andmedi nii tähestikus "ISO-8859-1" kui ka "UTF-8", kuid lugeda saab neid esialgu vaid tähestikus "ISO-8859-1".
- Versioonis 1.41 parsiti OCSP kehtivuskinnitus lahti ja hoiti mälus vaid sellest loetud vajalikke andmeid. Faili kirjutati algse OCSP kehtivuskinnituse räsi aga kehtivuskinnitus ise koostati uuesti. See rikkus ära räsi väärtuse. Alates versioonist 1.42 hoitakse mälus kogu algset OCSP kehtivuskinnitust ja salvestatakse muutmata kujul ka faili. Räsi vastab faili salvestatud kehtivuskinnitusele.
- Parandatud base64 formadis binary objektide parsimist ja parandatud osad vead seoses ebastandardse reapiikkusega objektide parsimisel.
- Lisatud AID kaartide toetus PKCS#11 ja CSP draiverite kasutamise puhul.
- Elemendil `<Reference>` mis sisaldab elemendi `<SignedProperties>` räsi peab olema atribuut: `Type="http://uri.etsi.org/01903/v1.1.1#SignedProperties"`.
- Elementidel `<SignedInfo>` ja `<SignedProperties>` võib, kuid ei pea olema atribuuti: `xmlns=http://www.w3.org/2000/09/xmldsig#`.
- Parandatud on xml faili parsimise kiirust. Nüüd loetakse kogu info failist ühe korraga ja ei ole vaja enam teist korda üksikuid elemente lugeda et nende täpset räsi arvutada. Elementide info koostatakse nüüd SAX sündmuste andmetest.

Erinevused teegi versioonide 1.52 ja 1.59 vahel

- Integreeriti FreeBSD fix
- Lisati veakontroll juurdepääsutõendi kehtivuse lõppemise puhul. Sellega seoses tekkis veel üks uus veakood.
- Lisati parameeter `bUseCA` funktsioonidele `verifySigDoc()`, `verifySignatureInfo()`, `verifySignatureInfoCERT()`, `verifySigDocCERT()` võimaldamaks jätta kontrollist välja CA serdi abil kontrollimine ja seega kontrollida ka selliseid dokumente, mille CA sertifikaati kontrollijal ei ole (võõraste CA-de toetus)
- Parandati vana ja uue teegi yhilduvust. Faili uuesti salvestamisel ei dekodeerita enam `DataFile` objektide sisu (base64), kuna sellega võib kaasneda whitespace kadu ja seega allkirja invalideerumine, vaid kopeeritakse terve `DataFile` sisu ajutisse faili ja sealt uude faili.
- Parandati null pikkusega faili allkirjastamise probleem.
- Lühendatid ajutiste failide nimesid. Nüüd koostatakse ajutised failid mustri alusel "ddocXXXXXX", kus viimased kuus kohta määrab süsteem nii et tekiks unikaalne failinimi.
- Ajutised failid salvestatakse süsteemi ajutiste failide kataloogi.
- Ajutisi faile luuakse nüüd vaid faili (ümber)salvestamisel kus see annab olulise võidu väiksema mälu vajadusena. Enam ei kasutata ajutisi faile XML

kanoniseerimisel kus see erilist võitu ei anna.

- Kontrolliti UTF-8 kujul failinimed kasutamist. Leiti, et sellisel kujul failinimesid ei suuda fopen() kasutada ja muud funktsioonid ei ole piisavalt laialt levinud. Otseselt süsteemi tähestikus (ISO-8859-1) failinimedega aga saab nii Windows kui Linux hakkama. Ka sellistega mis sisaldavad täpitähti. Linuxi puhul on vajalik uuem 2.4 kerneliga versioon. Osades vanemates versioonides võib probleeme tekkida.
- Kontrolliti BIO paketi rakendatavust fopen() asemel ja leiti et pole mõtet kuna BIO ise seda funktsiooni kasutab ja ei ole seega loodetuks porteeruvaks variandiks.
- Lisati struktuur PolicyIdentifier ja funktsioonid readCertPolicies(), PolicyIdentifiers_free() ja isCompanyCPSPolicy() asutuse sertifikaatide sertifitseerimispoliitikaga seotud atribuutide lugemiseks.
- Täiustati DigiDoc portaali utiliite
- Parandati funktsioone verifyNotaryInfo(), verifyNotaryInfoCERT() ja getConfirmation() täiustamaks kehtivuskinnituse kontrolli tundamtu CA puhul.

Juhised enam esinevate probleemide kiireks lahendamiseks

Kuidas luua uusi DigiDoc'e

Siin on lihtne ülevaade kuidas saaks luua DigiDoc'e C- teegi funktsioonidega. Kõigepealt tuleb defineerida tarvilikud struktuurid:

```
SignedDoc* pSigDoc;
```

See struktuur kajastabki DigiDoc'i formaati, k ik teised paiknevad siin sees.

```
DataFile* pDataFile;
```

Üks selline struktuur vastab igale DigiDoc'i kontainerile lisatud andmefailile. Ühes DigiDoc'is võib olla mitu andmefaili, kuid ilma andmefailita pole ka DigiDoc'i. Andmefail ise võib olla DigiDoc'i sisse lisatud aga võib jääda ka lisamata.

K i gepealt tuleb initsialiseerida teek:

```
initDigiDocLib();
```

See tagab kõigi kasutatavate OpenSSL'i väärtuste initsialiseerimise. Seejärel tuleb mälusse luua digidoc'i struktuur:

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

Nende kahe string konstanti väärtused on vastavalt "DIGIDOC-XML" ja "1.3" (DigiDocLib.h). Toetatud versioon 1.0, erinevus kahe versiooni vahel on

peamiselt andmefaili MIME tüübi erinevas kuvamises, kasutaja jaoks pole selle erinevuse teadmine vajalik, sest teegi funktsioonid teavad ise kuidas antud versiooni jaoks struktuure luua ja salvestada. Vea korral tagastatakse veakood. Versioonid 1.1 ja 1.2 ei erine üksteisest faili formaadi osas. Versioon 1.2 loodi peale oluliste vigade leidmist versiooniga 1.1. kaasas olnud libxml2 teegi vanemas versioonis. Vead esinesid ennekõike halja XML sisestamisel. Kui kasutate haljast XML-i siis kasutage kindlasti ka versiooni 1.2 juhendamaks kasutajat tõmbama endale uusimat tarkvara millega antud faili korrektselt lugeda saab. Versioonis 1.3 on parandatud olulised erinevused XAdES standardist.

Peale DigiDoc'i struktuuri loomist saab sinna lisada andmefaili(d):

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

Siin teiseks parameetriks on pointer DigiDoc'i struktuurile kuhu lisatakse antud andmefail. Kolmandaks parameetriks on andmefaili antud Digidoc'i piires unikaalne id (const char*), on soovitatav see parameeter jätta määramata, siis hoolitseb teek ise selle genereerimise eest. Neljandaks parameetriks on viide faili nimele, viiendaks andmefaili sisaldumisviis, siin on defineeritud järgmised väärtused (DigiDocLib.h):

- 1) CONTENT_DETACHED
- 2) CONTENT_EMBEDDED
- 3) CONTENT_EMBEDDED_BASE64

Esimene tähendab, et andmefaili sisu DigiDoc'i sisse ei lisata, ainult viited, soovitatav kui tegu on suurte failidega. Teine tähendab seda andmefaili sisu kopeeritakse otse DigiDoc'i sisse, seda on soovitatav kasutada ainult siis kui olete kindel, et andmefaili sees pole eritähendusega sümboleid, mis võiksid faili lugevatele ja kuvavatele programmidele ebaseeldivusi põhjustada, alati on kindlam seda võimalust mitte kasutada. Kolmas ning kõige kindlam ja lihtsam viis lisab kogu andmefaili, olles selle enne konverteerinud base64 kodeeringusse, üldjuhul on see kige soovituim. Faili nimi tuleks anda täieliku nimena koos teekonnaga antud alamkataloogini. Salvestamisel teekond faili nimest eemaldatakse kuid digidoc faili koostamisel on see vajalik, sest te võite lisada faile mitmest eri alamkataloogist. Funktsiooni viies parameeter on andmefaili MIME tüüp näiteks "application-x/ms-word" või "application-x/acrobat-pdf" või mis iganes sõltuvalt andmefailist.

Järgmised neli parameetrit peaksid tavajuhul jääma teegi enda arvutada, need on vastavalt:

- 1) andmefaili suurus baitides
- 2) andmefaili räsi
- 3) andmefaili räsi pikkus
- 4) andmefaili räsi tüüp (praegu lubatud ainult sha1)

Eelviimaseks parameetriks on sisestatavate andmete kooditabeli nimi. Teek toetab hetkel kahte tähestikku: ISO-8859-2 ja UTF-8. Mälus hoitaks andmeid UTF-8 tähestikus ja faili salvestatakse samuti UTF-8-s. Nende kooditabelite jaoks on defineeritud konstandid:

- CHARSET_ISO_8859_1

- `CHARSET_UTF_8`

Viimaseks parameetriks on faili nime kooditabel. Win32 keskkonnas tuleb üldjuhul edastada faili nimi ISO-8859-1 kodeeringus et teek seda faili avada saaks.

Muu kooditabeli nime kasutamisel tekib viga

Peale tagasitulekut sellest funktsioonist on meil antud DigiDoc'i kopeeritud andmefaili andmed ja sõltuvalt välja kutsuja soovist ka fail ise. Olgu öeldud, et viimase nelja parameetri väärtust see funktsioon ei arvuta, küll aga leiab unikaalse id kui seda polnud ette antud. Vea korral tagastatakse NULL.

Järgmisena arvutataksegi välja need suurused mida struktuuri loov funktsioon `DataFile_new()` ei teinud.

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

See funktsioon arvutab ja lisab need neli väärtust esimeseks parameetriks antud DigiDoc'i DataFile'i sektsiooni `pDataFile->szId`, arvutamiseks kasutatakse kolmandas parameetris antud faili nime.

Ainus lubatud räsi tüüp on `DIGEST_SHA1`. Funktsioon tagastab täisarvu, kui vigu ei olnud, siis on täisarvu väärtuseks `ERR_OK` (defineeritud `DigiDocLib.h` 's). Vea korral tagastatakse veakood.

Antud veakoodile vastavat veateate saab küsida funktsiooniga,

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

kus `rc` on veakood; veateated esitatakse inglise keeles.

On võimalik lisada ka suvalist infot antud andefaili kohta. Selleks kasutatakse funktsiooni `addDataFileAttribute()`. Näiteks:

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

Esimeseks parameetriks on viit andmefaili struktuurile, millele järgneb atribuudi nimi ja väärtus. Andmed tuleb edastada UTF-8 kodeeringus.

Digidoc'i faili loomiseks ja salvestamiseks kasutatakse järgmist funktsiooni:

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

See funktsioon saab parameetriteks viida salvestatavale digidoc'i struktuurile ning kaks faili nime, `oldfile` mis võib ka puududa, ning `output` ehk väljund faili nimi. Kui old fail on olemas ja teda õnnestub avada, siis kopeeritakse sealt andmefaili sisu.

Peale töö lõppu tuleb vabastada mälu, selleks kasutatakse funktsiooni

```
SignedDoc_free(pSigDoc);
```

mis vabastab ka kõigi andmefailide mälu.
Viimaseks ülesandeks on teegi sulgemine.
`finalizeDigiDocLib();`

Kuidas lisada kehtivuskinnitust ?

Funktsioon `getConfirmation()` loob Notari (edaspidi nimetatud OCSP - Online Certificate Status Protocol) päringu antud DigiDoc'i *pSigDoc* allkirjast *pSigInfo* , vastavalt funktsiooni parameetritle allkirjastab päringu kui see oli n utav ning saadab päringu notaryURL-i poolt osutatud aadressile.

Saadud vastus lisatakse DigiDoc'i *pSigDoc* base64 kodeeritult. Funktsiooni eduka täitmise korral tagastatakse ERR_OK vastasel juhul veakood.

Selgituseks olgu öeldud, et OCSP päringuga küsitakse, kas mingi sertifikaat on antud hetkel kehtiv.

OCSP päringusse pannakse küsitava sertifikaadi ja selle sertifikaadi välja andnud sertifikaadi andmed. Päring võib olla allkirjastatud.

```
int getConfirmation(SignedDoc* pSigDoc, SignatureInfo* pSigInfo,
                    const X509** caCerts, const X509* pNotCert,
                    char* pkcs12FileName, char* pkcs12Password,
                    char* notaryURL,
                    char* proxyHost, char* proxyPort)
```

Funktsiooni parameetrid:

- *pSigDoc* - uuritava digidoc'i struktuur, see peab juba eelnevat olema täidetud.
- *pSigInfo* - uuritava sertifikaadi poolt antud allkiri, ka see ei tohi puududa
-
-
-
- *pkcs12FileName* – OCSP-päringu teostamiseks vajaliku pääsutõendi asukoht.
- *pkcs12Password* - pääsutõendi kasutamiseks vajalik paroolifraas.
- *notaryURL* - URL OCSP teenuse pakkujale. Parameeter ei tohi puududa.
- *proxyHost* - kui *notaryURL*-i kätte saamiseks on tarvilik kasutada proksit peab see parameeter olema täidetud, vastasel juhul peab olema NULL.
- *proxyPort* - kui *notaryURL*-i kätte saamiseks on tarvilik kasutada proksit peab see parameeter olema täidetud, vastasel juhul peab olema NULL.

Näide:

Küsi kinnitust digidoci "c:\\tt.ddoc" allkirjale "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");
```

Kuidas otsida sertifikaate DigiDoc'i funktsioonides ?

Funktsioon vajab kolme sertifikaati : allakirjutanud serdi välja andja, OCSP ehk Notari ning OCSP päringu allkirjastaja oma. Kui OCSP päringut ei allkirjastata siis viimast vaja ei ole.

Serdid on saadavad kolmel eri viisil: MS CertStore, PKCS#12 konteiner ja tavaline PEM fail (edaspidi viidatud kui X509 fail).

Sertide otsimiseks kasutatakse struktuuri CertSearch_st kus muutuja searchType määrab ära otsimisviisi.

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

Muutujal searchType võib olla kolm väärtust:

CERT_SEARCH_BY_STORE

CERT_SEARCH_BY_X509

CERT_SEARCH_BY_PKCS12

Kui valitakse CERT_SEARCH_BY_STORE siis peab olema struktuuris täidetud ka certSearchStore teised liikmed võivad puududa.

Kui valitakse CERT_SEARCH_BY_PKCS12 siis peavad struktuuris olema täidetud ka pkcs12FileName ja pswd teised liikmed võivad puududa.

Kui valitakse CERT_SEARCH_BY_X509 s.t sertifikaat on PEM failis siis peab struktuuris olema täidetud x509FileName. Kui antud sertifikaati kasutatakse ka allkirjastamiseks siis peavad lisaks x509FileName olema täidetud ka keyFileName ja pswd.

Struktuuri CertSearchStore_st kasutatakse sertide otsimiseks MS CertStore'st.

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

MS CertStore'st saab sertifikaate praegu otsida kolmel eri viisil: seerianumbri , subjekti ja väljaandja nime järgi, ning neid otsimisviise saab vastavalt soovile

omavahel kombineerida. Näiteks saab otsida sertifikaati mis oleks ette antud seerianumbri, välja andja nime ja subjekti nimega, või etteantud seeria numbri ja väljaandjaga.

Otsimisviisi konstandid on:

```
CERT_STORE_SEARCH_BY_SERIAL  
CERT_STORE_SEARCH_BY_SUBJECT_DN  
CERT_STORE_SEARCH_BY_ISSUER_DN  
CERT_STORE_SEARCH_BY_KEY_INFO
```

Ja neid võib searchType'ks kokku OR'ida.

Kui otsingu tüüp sisaldab otsimist seerianumbri järgi siis tuleb seerianumber kirjutada väljale certSerial. Veidi keerulisem on ette anda otsingu parameetreid subjekti ja issueri jaoks.

Otsingu parameetriteks on stringid millest igaüks peab olema nõutud väljal (vastavalt kas subjekti või issueri väljal), kokku moodustavad need siis kas subDNCriterias vi issDNCriterias massiivi, selle suurus (ehk parameetrite arv) peab olema kirjas vastavalt numberOfSubDNCriterias vi numberOfIssDNCriterias muutujas. Parameeter stringid ei ole tõstutundlikud (enne võrdlemist teisendatakse kõik tähed suurteks) ja tühikute arv ei oma samuti tähtsust (need eemaldatakse enne võrdlemist).

Näiteks olgu vaja leida MS CertStorest EST-EID sertifikaat (kõigi Eesti ID kaartide sertide väljaandja). Tegutsemissskeem on sel juhul põhimõtteliselt järgmine:

```
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;
```

Kuidas avada ja lugeda DigiDoc'e

Vaatame näitekoodi, kust parema ülevaate saamiseks on veakontrollid, väheoluliste muutujate deklaratsioonid jms välja jäetud.

```
SignedDoc* pSigDoc;  
DataFile* dataFile;  
SignatureInfo* pSigInfo;  
  
initDigiDocLib();  
rc = readSignedDoc(&pSigDoc, infile, 0, buf);  
andmefailideArv=getCountOfDataFiles(pSigDoc);  
allkirjadeArv=getCountOfSignatures(pSigDoc);  
for(counter=0;;counter++){
```

```

        //kui sellise järjekorra numbriga andmefaili pole
        // saada tagasi NULL'i
        dataFile = getDataFile(pSigDoc,counter);
        if(!dataFile){
            break;
        }
        // teeme midagi andmefaili struktuuriga
    }
    for(counter=0;counter< allkirjadeArv;counter++){
        pSigInfo=getSignature(pSigDoc,counter);
        // teeme midagi allkirja struktuuriga
    }
    SignedDoc_free(pSigDoc);
    finalizeDigiDocLib();

```

Digidoc faili avamiseks ja lugemiseks kasutatakse funktsiooni `readSignedDoc()`. Selle funktsiooni parameetriteks on digidoc'i viite aadress kuhu kirjutatakse loetud digidoc'i struktuur. Teiseks parameetriks on Digidoc'i faili nimi, kolmas on lipp mis näitab, kas on vaja andmefaili räsi kontrollimine lugemise ajal on nõutud. Kui andmefaili pole lisatud peab see kindlasti olema `false`. Neljandaks parameetriks on puhver, kuhu XML'i parsiv SAX parser saab kirjutada oma veateateid.

Teised funktsioonid lihtsalt näitavad DigiDoc'i struktuuri sisu.

Kuidas lisada allkirja DigiDoc'ile

Vaatame näitekoodi, kust parema ülevaate saamiseks on weakontrollid, väheoluliste muutujate deklaratsioonid jms. on välja jäetud.

```

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();

```

Kõigepealt tuleb digidoc'i struktuur luua või lugeda failist. Neid tegevusi

kirjeldatakse eraldi dokumentides, siin loetakse digidoc'i struktuur failist funktsiooniga `readSignedDoc()`.

Järgmisena tuleb luua `SignatureInfo` struktuur funktsiooniga `SignatureInfo_new()` esimeseks parameetriks on viit uut struktuuri hoidvale viidale, teisesks antud digidoc ise, kolmandaks allkirja info unikaalne nimi, on soovitatav seda mitte pakkuda, sel juhul arvutab teek selle ise. Funktsioon `addAllDocInfos()` lisab loodud allkirja info struktuuri digidoci struktuuri. Funktsioon `setSignatureProductionPlace()` lisab andmed allkirjastamise asukoha kohta. Funktsioon `addSignerRole()` lisab allkirjastaja rolli allkirja info struktuuri `pSigInfo`, teine parameeter näitab kas roll on kinnitatud või mitte (0-kinnitamata/1-kinnitatud), kolmas parameeter on rolli nimi, neljas rolli nime pikkus – 1 tähendab, et teek proovib selle ise leida. Eelviimane parameeter näitab et, kas rolli nimi tuleb panna base64 kodeeringusse (1) või mitte (0). Viimane parameeter on sisendandmete kooditabel.

Allkirjastamist läbi MS Crypto Service Provideri teostab funktsioon `calculateSigInfoSignatureWithCSPEstID()` allkirjastada võib ka PKCS#11 kui teie kaardilugeja seda toetab ja vastav teek on installeeritud, sel juhul kasutatakse funktsiooni `calculateSignatureWithEstID()`. Need funktsioonid lisavad arvutatud allkirja ka kohe digidoc'i struktuuri.

Lõpuks tuleb uuendatud struktuur salvestada funktsiooniga `createSignedDoc()`.

Näiteprogrammid DigiDoc'i C-teegi kasutamiseks

Järgnevad mõningad teegi kasutamist illustreerivad näited, mida saab otse kopeerida.

CreateDoc.c

Seda näiteprogrammi on proovitud nii LINUXil kui Windows platvormil. Programm tekitab ja salvestab uue DigiDoc'e struktuuri kasutades ette antud andmefaili ja MIME tüüpi. Sisend parameetriteks ongi kolm suurust :

- andmefaili nimi
- andmefaili MIME tüüp
- failinimi kuhu salvestatakse uus DigiDoc

```
#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

See näite programm demonstreerib allkirjade lugemist, nende näitamist ja kontrollomist. Programmi käivitamiseks on tarvilikud parameetrid:

1. `infile` = Ühte või enam allkirja sisaldav DigiDoc'i fail.
2. `cafile` = Allkirja(de) väljaandja sertifikaat.
3. `rootca` = Juursertifikaat
4. `notcert` = Notari (OCSP) sertifikaat.
5. `capath` = Kataloog täiendavate sertifikaatide lugemiseks, kui rohkem sertifikaate ahelas ei ole võib selle parameetri väärtus olla suvaline.

```

#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; // DETACHED, 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 atribuutide arv
    char* szCharset;           // datafile initial
    codepage
    char** pAttNames;           // atribuutide nimed
    char** pAttValues;          // atribuutide 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 atribuudid 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 atribuudid

```
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 kasutatakse 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; // Privaatvõrme faili nimi
char* pkcs12FileName; //PKCS #12 konteineri nimi
char * pswd; // privaetvõrme/ PKCS #12 parool
CertSearchStore* certSearchStore; //MS sertifikaatide
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 seerinumbr
    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; //vääraaks osutunud kontrollavaldis
} ErrorInfo;

```

CSPProvider

Hoiab teegi jaoks vajalikke CSP parameetreid.

```

typedef struct CSPProvider_st {
    char* CSPName;
    int rsa_full; // kui FALSE, siis kasutatakse RSA_SIG
    int at_sig; // kui FALSE, siis kasutatakse AT_KEYEXCHANGE
} CSPProvider;

```

CertItem

Üldine struktuur X509 viitade nimistu hoidmiseks:

```

typedef struct CertItem_st {
    X509* pCert;
    struct CertItem_st* nextItem;
} CertItem;

```

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 funktsioon 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 millest 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 osutada.

## 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 salvestatakse 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 algandmete 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 puhvrise 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);
```

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 nähtud, peab siis olema 0.

- newDataFile - loodav struktuur.
- pSigDoc - allkirjastatud dokumendi struktuuri aadress
- id - algandmefaili unikaalne tunnus. Kasuta NULL-i vaikimisi 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 atribuudi

- pDataFile - algandmefaili info aadress
- name - atribuudi nimi
- value - atribuudi väärtus

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

getDataFileAttribute()

Tagastab algandmefailile soovitud atribuudi

- pDataFile - algandmefaili info aadress
 - idx - atribuudi indeks
 - name - puffer atribuudi nime aadressi jaoks
 - value - puffer atribuudi 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);
```

### 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 stukturi 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);
```

### 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);
```

### calculateSignatureWithEstID()

Arvutab SHA1+RSA allkirja väärtuse ja salvestab ta antud allkirja infos. See on EstID allkiri. Antud funktsioon on hetkel Win32 spetsiifiline, vajab vastavat EstID teeki, PKCS#11 ohjurprogrammi ja kaardilugejat!!!

- pSigDoc - allkirjastatv digidoc
- pSigInfo - allkirja info
- slot - kaardi "sloti" ehk võtmepaari indeks. EstiID kaardil onnäiteks autentimise slot 0 ja allkrijastamise slot 1. Seda aga oletatavasti vaid EYP ohjurprogrammi puhul!!!
- passwd - valitud "sloti" salajase võtme PIN (s.o. PIN2)

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

### calculateSigInfoSignatureWithCSPEstID()

Arvutab SHA1+RSA allkirja väärtuse ja salvestab ta antud allkirja infos. Siit tuleb samasugune EstID allkiri kui eelmisest ainult, et siin eeldatakse arvutis EstEID MS Crypto Service Provider'i olemasolu. Vastavad ohjurprogrammid saab laadida firma ITS kodulehelt.

pSigDoc - allkirjastatv digidoc  
pSigInfo - allkirja info

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

### GetSignParametersWithEstIdCSP()

Üldine funktsioone allkirjade ja räside arvutamiseks ESTEID kaardi abil läbi MS Crypto Service Provider'i, selle funktsiooni abil loetakse ka ESTEID kaardi sertifikaat Windowsi sertifikaatide hoidlast.

- dataToBeSigned - sisend andmed
- dataLen - sisend andmete pikkus
- cert - puhver allkirjastaja sertifikaadi tagastamiseks, kui

```

sertifikaati ei vajata v  b olla NULL
• certLen - tagastatud sertifikaadi pikkus
• keyBlob - allkirjastatud privaat v  me avaliku paarilise puhver,
kui seda ei vajata v  b olla NULL
• keyBlobLen - tagastatud avaliku v  tme pikkus
• hash - tagastatud r  si puhver, kui seda ei vajata v  b olla NULL
• hashLen - tagastatud r  si pikkus
• sign - allkirja puhver
• sigLen - tagastatud allkirja pikkus
int GetSignParametersWithEstIdCSP(byte * dataToBeSigned,unsigned long
dataLen, X509 **x509, int *needCert,byte *keyBlob, unsigned long
keyBlobLen, byte *hash, unsigned long *hashLen,byte *sign,unsigned
long *sigLen);

```

### 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);

```

### NotaryInfo\_new()

Allokeerib m  lu 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 lugemine ja kirjutamine***

Selle kategooria funktsioonide abil saab allkirjastatud dokumente lugeda, kirjutada ja algandmefaile eraldi faili salvestada.

### createSignedDoc()

Salvestab allkirjastatud dokumendi soovitud faili

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

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

## readSignedDoc()

Loeb allkirjastatud dokumendi antud failist

- ppSigDoc - puffer allokeeritud dokumendi info jaoks
- szFileName - sisendfaili nimi
- checkFileDigest - (0/1) kas kontrollida viidatavate andmefailide räsikoodi.
- errbuf - puffer veatekstide jaoks

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

## extractDataFile()

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

- szFileName - sisendfaili nimi
- szDataFileName - väljundfaili nimi
- szDocId - soovitud algandmefaili tunnus
- szCharset - soovitud väljundi kooditabel
- szFileNameCharset - faili nimes kasutatud kooditabel

Funktsioon tagastab ERR\_OK või veakoodi.

```
int extractDataFile(const char* szFileName,
 const char* szDataFileName,
 const char* szDocId, const char* szCharset,
 const char* szFileNameCharset)
```

## ***MS sertifikaadihoidla funktsioonid***

Need funktsioonid oskavad suhelda MS sertifikaadihoidlaga.

### CSPProvider\_new ()

Loob ja tagastab uue CSPProvider struktuuri instantsi.

```
CSPProvider * CSPProvider_new();
```

### CSPProvider\_free()

Vabastab CSPProvider elemendi.

- cCSPProvider - CSPProvider element
- ```
void CSPProvider_free(CSPProvider * cCSPProvider);
```

isActiveCSPProvider()

Kontrollib, kas CSP on aktiivne.

- cCSPProvider - viit CSP infole
- ```
int isActiveCSPProvider(CSPProvider * cCSPProvider);
```

### getCurrentCSPProvider ()

Tagastab hetkel teegis kasutusel oleva CSP info või nulli, kui otsing ebaõnnestus.

- tryToFindIfMissing - FALSE korral ei hakka CSP-d otsima.
- ```
const CSPProvider * getCurrentCSPProvider(int tryToFindIfMissing);
```

setCurrentCSPProvider ()

Määrab teegis hetkel kasutatavaks uue CSP.

- newProvider - viit CSP infole
- ```
void setCurrentCSPProvider(CSPProvider * newProvider);
```

### findIssuerCertificatefromMsStore()

Otsib sertifikaadi väljaandjat MS sertifikaadihoidlast. Tagastab väljaandja sertifikaadi või nulli, kui seda ei leitud. Otsib läbi hoidlad "My", "CA" ja "Root".

```
X509 * findIssuerCertificatefromMsStore(X509 *x509);
```

### resolveCertChain\_ByCertStore ()

Koostab ja valideerib sertifikaatide ahela, kui see eksisteerib.

- pListStart - viit CertItem struktuurile, kuhu on talletatud analüüsitav sertifikaat. Võib viidata suvalise astme sh. juur-sertifikaatidele.

```
int resolveCertChain_ByCertStore(CertItem* pListStart);
```

## **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 sertifikaatide 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 räsied 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 atribuutide info
- szFileName - algandmefaili nimi DETACHED tüüpi 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 atribuutide 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
- CPath - 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 *CPath, 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
- CPath - sertifikaatide kataloogi nimi
- notCertFile - kehtivuskinnituse andja sertifikaat

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

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 enamasti 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 struktuurist)

```
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_DETACHED	"DETACHED"
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 mälu puhvri 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
	// Ebakorrektn 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_OSSL_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 v ti	
// 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"