JDigiDoc Teegi Kasutusjuhend

Veiko Sinivee
S|E|B IT Partner Estonia

Sisukord

JDigiDoc

JDigiDoc on programeerimiskeeles Java loodud teek. Antud teek pakub funktsionaalsust DIGIDOC-XML 1.3, 1.2 ja 1.1 formaadis digitaalselt allkirjastatud failide loomiseks, lugemiseks, allkirjastamiseks, kehtivuskinnituse hankimiseks ja allkirjade ning kehtivuskinnituste kontrolliks.

JDigiDoc klassid järgivad küllaltki täpselt XML-DSIG ja ETSI standardit ja pakuvad mugavat kasutajaliidest antud objektide loomiseks ja kasutamiseks. Antud dokument annab ülevaate JDigiDoc teegi arhitektuurist, konfigureerimisest ja kasutamisest.

Teegi loomisel on võetud eesmärgiks tagada toimivus võimalikult mitmesugustes keskkondades ja võimaldada kasutajal ebasobivaid teegi osasid teiste sama funktsionaalsust pakkuvate osade vastu välja vahetada. Selle saavutamiseks on JDigiDoc loodud puhta Java teegina mis ei eelda näiteks toimimist J2EE keskkonnas kuid võib ka antud keskkonnas toimida. Paindlikuse saavutamiseks on olulisemad algoritmid kogutud vastavat funktsionaalsust pakkuvatesse Factory klassidesse mida kasutatakse antud funktsionaalsust defineeriva Interface kaudu. Seega on kasutajal alati võimalik antud Factory teise sama interfacet täitva klassi vastu välja vahetada.

DigiDoc dokument on esitatud XML kujul ning põhineb rahvusvahelistel standarditel XML-DSIG ja ETSI TS 101 903.

Viited

- 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 Digitaalallkirja Seadus
- DigiDoc formaat - DigiDoc failide vorming. http://www.id.ee/files/digidoc_vorming_1_3.pdf

Sõltuvused

JDigiDoc teek sõltub järgmistest komponentidest:

- Java2 – JDK/JRE 1.3.1 või uuem
- Apache XML Security – Vajalik kanoniseerimiseks
- XML parser – Apache Xerces. Vajalik Apache XML Security teegi jaoks. Paraku ei saa kasutada mingit muud XML parserit nagu JAXP, sest Apache XML Security kasutab ainult seda parserit. Muidugi saaks DigiDoc failide lugemisel kasutada siiski muud parserit aga sel pole vist mõtet.
- Xalan – Versioon 2.2D13 või uuem. Vajalik Apache XML Security teegi jaoks
- IAIK JCE krüptoteek – Vajalik IAIKNotaryFactory klassis kehtivuskinnituste koostamiseks ja parsimiseks. Kui see factory välja vahetada siis pole IAIK JCE teeki vaja.

- Bouncy-Castle krüptoteek – Vajalik PKCS11DigiDocFactory klassis krüptograafilisteks operatsioonideks. Sobiks tegelikult suvaline Java krüptoteek. Bouncy-Castle teek valiti kuna ta on vabavara teek.
- Jakarte Log4j - Vajalik Apache XML Security teegi jaoks

Töökeskkond

JDigiDoc kasutamiseks tuleb lisada oma arendusümbruskonna CLASSPATH -i järgmised teegid:

- JDigiDoc.jar – JDigiDoc teek
- jce-jdk13-114.jar – Bouncy-Castle Java krüptoteek. Võib ka uuem versioon või mõni teine krüptoteek.
- iak_jce.jar – IAIK Java krüptoteek. Ainult juhul kui kasutatakse IAIKNotaryFactory -t
- jakarta-log4j-1.2.6.jar - Jakarte Log4j teek
- xmlsec.jar – Apache XML Security teek
- xalan.jar, xercesImpl.jar, xml-apis.jar, xmlParserAPIs.jar – Xalan ja Xerces

Kui soovite kasutada RSA-SHA1 allkirja väärtuse arvutamiseks ID kaardi abil PKCS#11 liidest, siis lisage CLASSPATH -i:

- iaikPkcs11Wrapper.jar

ja kopeerige mingisse PATH ümbruskonnamuutujas loetletud kataloogidest (näiteks SYSTEM32) järgmised DLL-d:

- esteid-pkcs11.dll
- Pkcs11Wrapper.dll

Linux keskkonnas tuleks kopeerida kataloogi {JAVA_HOME}\jre\lib\i386 failid:

- libesteid-pkcs11.so
- libpkcs11wrapper.so

Konfigureerimine

JDigiDoc kasutab oma konfiguratsiooni lugemiseks klassi ee.sk.utils.ConfigManager. Antud klass loeb konfiguratsiooniandmed Java property failist nimega JDigiDoc.cfg. Antud fail on tavaliselt JDigiDoc.jar sees, kuid võib olla ka suvalises muus kohas, millele viitab CLASSPATH. Võib salvestada ka mujale ja anda meetodile ConfigManager.init() faili täielik nimi.

Konfiguratsioonifailis on järgmised kirjed:

- Factory klasside valik. Kui asendate mõne Factory omalooduga, siis piisab selle klassi registreerimisest siin ja CLASSPATH-i lisamisest.

```
DIGIDOC_SIGN_IMPL=ee.sk.digidoc.factory.PKCS11SignatureFactory
DIGIDOC_NOTARY_IMPL=ee.sk.digidoc.factory.IAIKNotaryFactory
DIGIDOC_FACTORY_IMPL=ee.sk.digidoc.factory.SAXDigiDocFactory
CANONICALIZATION_FACTORY_IMPL=ee.sk.digidoc.factory.DOMCanonicalizationFactory
CRL_FACTORY_IMPL=ee.sk.digidoc.factory.CRLCheckerFactory
```

- Java krüptoteekide valik.

```
# Security settings
DIGIDOC_SECURITY_PROVIDER=org.bouncycastle.jce.provider.BouncyCastleProvider
IAIK_SECURITY_PROVIDER=iaik.security.provider.IAIK
```

- PKCS#11 seadistused

```
# EstID kaardi ohjurprogramm
DIGIDOC_SIGN_PKCS11_DRIVER=esteid-pkcs11
# AID kaardi ohjurprogramm
#DIGIDOC_SIGN_PKCS11_DRIVER=pk2priv
DIGIDOC_SIGN_PKCS11_WRAPPER=pkcs11rapper
DIGIDOC_VERIFY_ALGORITHM=RSA/NONE/PKCS1Padding
```

- Kehtivuskinnituse seadistused

```
DIGIDOC_DRIVER_BASE_URL=http://localhost:8080/XMLSign/
DIGIDOC_OCSP_RESPONDER_URL=http://ocsp.sk.ee
DIGIDOC_PROXY_HOST=<teie proxy serveri nimi>
DIGIDOC_PROXY_PORT=<teie proxy pordi number>
SIGN_OCSP_REQUESTS=true (või “false” kui ei soovi OCSP päringuid allkirjastada)

# Kehtivuskinnituste serveri juurepääsutõend
DIGIDOC_PKCS12_CONTAINER=<teiele SK poolt antud PKCS#12 faili nimi ja asukoht>
DIGIDOC_PKCS12_PASSWD=<faili salasõna>
DIGIDOC_OCSP_SIGN_CERT_SERIAL=<teile väljastatud sertifikaadi number>

# OCSP responderi sertifikaadtide nimed ja asukohad
DIGIDOC_OCSP_COUNT=2
DIGIDOC_OCSP1_CN=ESTEID-SK OCSP RESPONDER
DIGIDOC_OCSP1_CERT=<kataloog-teie-systeemis>\\esteid-ocsp.pem
DIGIDOC_OCSP1_CA_CERT=<kataloog-teie-systeemis>\\esteid.pem
DIGIDOC_OCSP1_CA_CN=ESTEID-SK
DIGIDOC_OCSP2_CN=KLASS3-SK OCSP RESPONDER
DIGIDOC_OCSP2_CERT=<kataloog-teie-systeemis>\\KLASS3-SK-OCSP.pem
DIGIDOC_OCSP2_CA_CERT=<kataloog-teie-systeemis>\\KLASS3-SK.pem
DIGIDOC_OCSP2_CA_CN=KLASS3-SK

# OCSP or CRL selectors
DIGIDOC_CERT_VERIFIER=OCSP
DIGIDOC_SIGNATURE_VERIFIER=OCSP
```

- Logimise seadistused

```
# log4j konfig faili asukoht MUUDA SEDA
DIGIDOC_LOG4J_CONFIG=C:\\JDigiDoc\\SignatureLogging.properties
```

- CA sertifikaatide asukohad

```
DIGIDOC_CA_CERTS=3
DIGIDOC_CA_CERT1=<kataloog-teie-systeemis>\\juur.pem
DIGIDOC_CA_CERT2=<kataloog-teie-systeemis>\\esteid.pem
DIGIDOC_CA_CERT3=<kataloog-teie-systeemis>\\KLASS3-SK.pem
```

- CRL seaded kui vaja

```
CRL_USE_LDAP=false
CRL_FILE=esteid.crl
CRL_URL=http://www.sk.ee/crls/esteid/esteid.crl
CRL_SEARCH_BASE=cn=ESTEID-SK,ou=ESTEID,o=AS Sertifitseerimiskeskus,c=EE
CRL_FILTER=(certificaterevocationlist;binary=*)
CLR_LDAP_DRIVER=com.ibm.jndi.LDAPCtxFactory
CRL_LDAP_URL=ldap://194.126.99.76:389
CRL_LDAP_ATTR=certificaterevocationlist;binary
CRL_PROXY_HOST=<teie proxy serveri nimi
CRL_PROXY_PORT=<teie proxy pordi number>
```

JDigiDoc kasutab XML Security teeki osaliselt XML kanoniseerimiseks. Paraku eeldab nimetatud teek et XML dokumentides on DTD viited ja trykib hulgaliselt hoiatusi kui seda ei leia. Selleks et antud probleemits lahti saada võib näitelks Log4j konfig failis peamise ehk default loggeri seadistada väga lakooniliseks ja siis selektiivselt valide need klassid mille kohta infot soovitakse. Näiteks:

```
# root logger properties
log4j.rootLogger=FATAL, output

# JDigiDoc loggers
```

```
log4j.logger.ee.sk.utils.ConfigManager=DEBUG, output
log4j.logger.ee.sk.digidoc.DigiDocException=DEBUG, output
log4j.logger.ee.sk.digidoc.factory.IAIKNotaryFactory=DEBUG, output
log4j.logger.ee.sk.digidoc.factory.SAXDigiDocFactory=DEBUG, output
log4j.logger.ee.sk.digidoc.factory.PKCS11SignatureFactory=INFO, output

#setup output appender
log4j.appender.output =org.apache.log4j.ConsoleAppender
log4j.appender.output.layout=org.apache.log4j.PatternLayout
log4j.appender.output.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%c{1},%
p] %M; %m%n
```

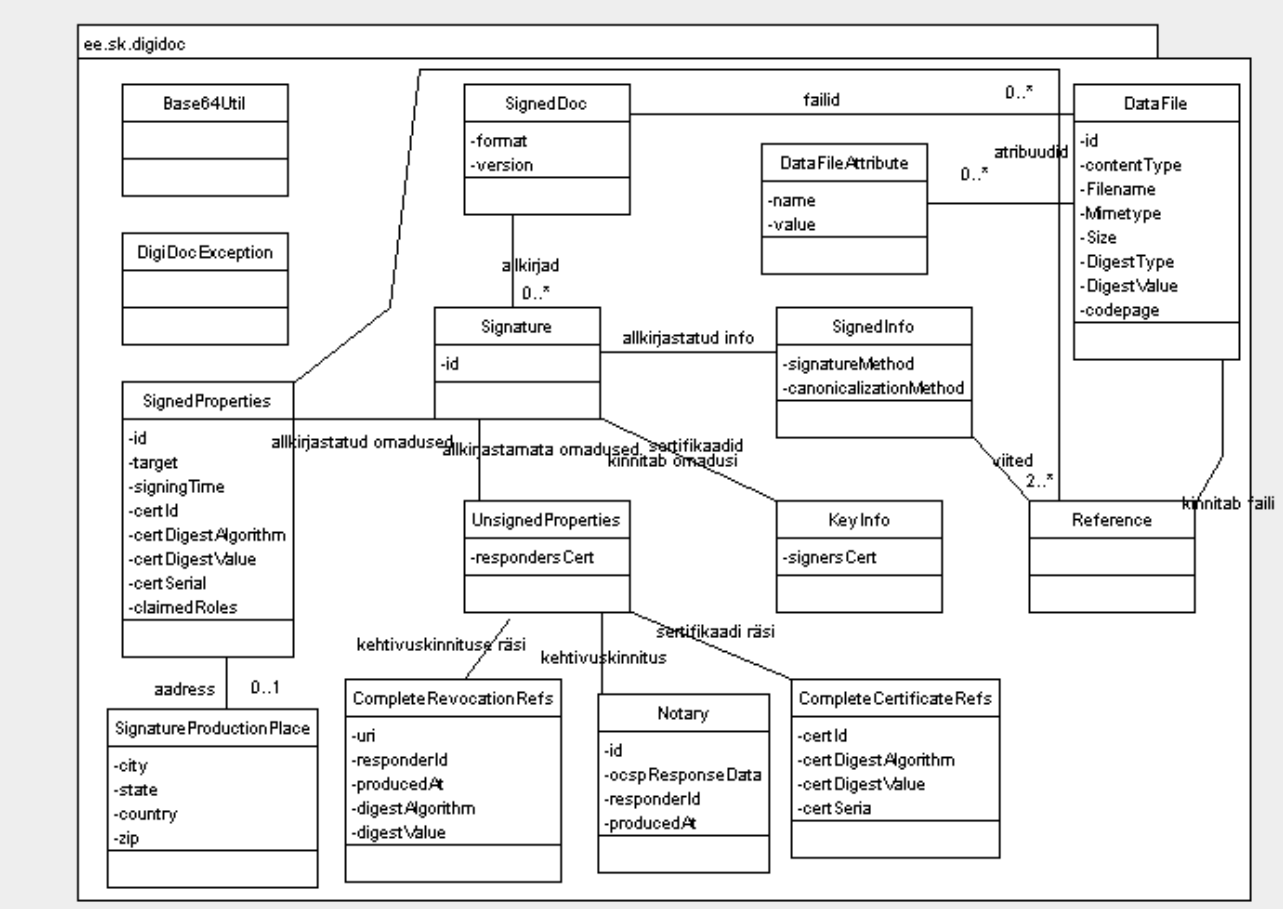
JDigiDoc arhitektuurist

JDigiDoc teek sisaldab järgmisi pakette:

- ee.sk.digidoc – Põhilised JDigiDoc klassid, mis järgivad DigiDoc failide XML elementide struktuuri.
- ee.sk.digidoc.factory – Sisaldab mitmesuguseid algoritme implementeerivaid klasse ja antud funktsionaalsust defineerivaid intefacesid.
- ee.sk.utils – Konfiguratsiooni ja muut utility klassid
- ee.sk.test – Näiteprogrammid

Pakett: ee.sk.digidoc

Selles paketis on põhilised JDigiDoc klassid, mis järgivad DigiDoc failide XML elementide struktuuri.

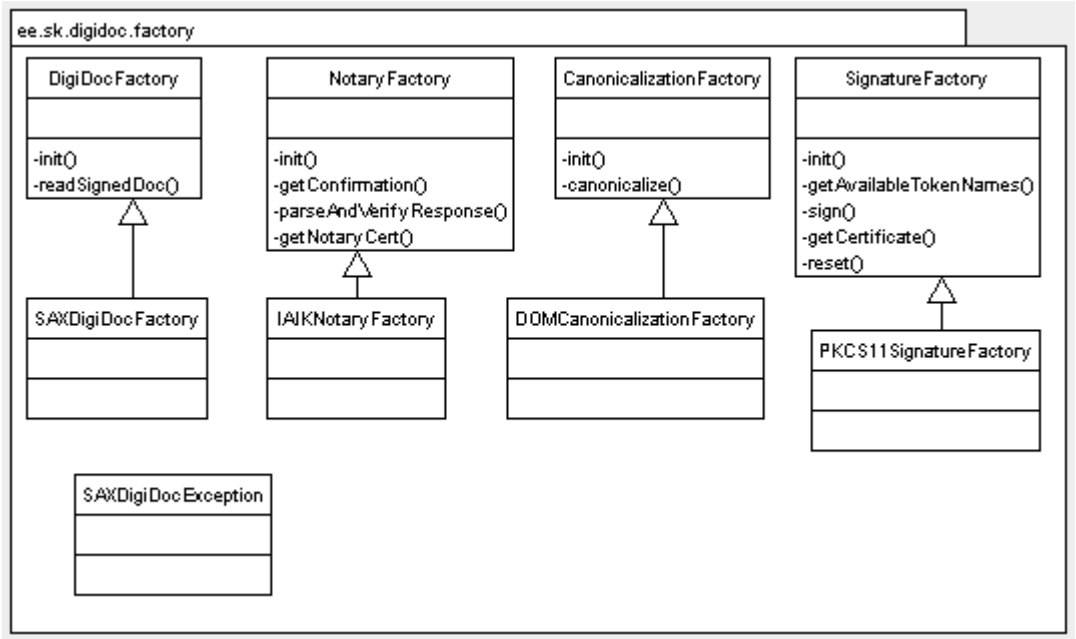


- ee.sk.digidoc.SignedDoc - Modelleerib DIGIDOC-XML 1.3, 1.2 või 1.1 formaadis digiallkirjastatud faili. Sisaldab andmefaile ja allkirju

- ee.sk.digidoc.DataFile – Allkirjastatud andmefaili andmed. Andmefaile saab lisada EMBEDDED_BASE64 kujul (binary failid), EMBEDDED kujul (puhas XML või tekst) ja DEATCHED kujul (viide välisele failile).
- ee.sk.digidoc.DataFileAttribute – Kasutaja lisatud andmefaili mittenõutud atribuudi andmed. Peale nõutud atribuutide (Id, Filename, Contenttype, MimeType, Size) ja formaadi poolt kinnitatud aga enamasti mittenõutud atribuutide (DigestType, DigestValue, Codepage) võib kasutaja lisada suvalisi muid atribuute.
- ee.sk.digidoc.Signature – Allkirja andmed
- ee.sk.digidoc.SignedInfo – allkirjastatud objetide viiteid sisaldav objekt.
- ee.sk.digidoc.Reference – Viide allkirjastatud objektile koos antud objekti räsikoodiga.
- ee.sk.digidoc.KeyInfo – Allkirjastaja sertifikaadi andmed
- ee.sk.digidoc.SignedProperties – allkirjastatud allkirja omadused
- ee.sk.digidoc.SignatureProductionPlace – allkirjastaja aadress.
- ee.sk.digidoc.UnsignedProperties – allkirja allkirjastamata omadused
- ee.sk.digidoc.Notary – kehtivuskinnitus
- ee.sk.digidoc.CompleteCertificateRefs – allkirjastaja sertifikaadi info ja räsikood.
- ee.sk.digidoc.CompleteRevocationRefs – kehtivuskinnituss info ja räsikood.
- ee.sk.digidoc.DigiDocException – spetsiaalne JDigiDoc exception klass.
- ee.sk.digidoc.Base64Util – Base64 dekodeerija ja kodeerija.

Pakett: ee.sk.digidoc.factory

Selles paketis on mitmesuguseid algoritme implementeerivad klassid ja antud funktsionaalsust defineerivad intefacesid.

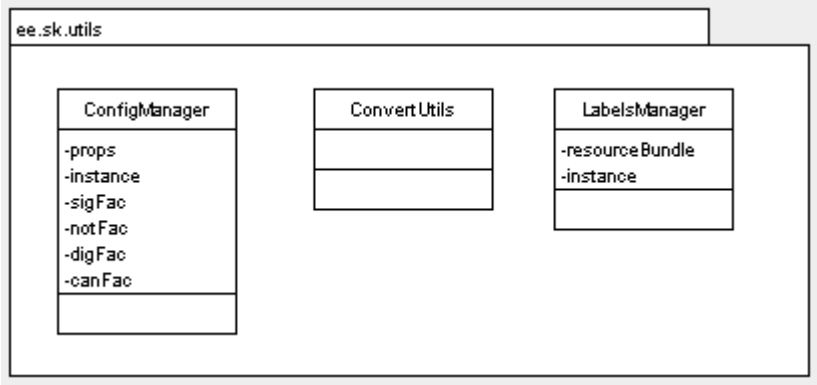


- ee.sk.digidoc.factory.DigiDocFactory – DigiDoc failide lugemist kirjeldav interface
- ee.sk.digidoc.factory.SAXDigiDocFactory – DigiDoc failide lugemise implementeering SAX parseri abil.
- ee.sk.digidoc.factory.SAXDigiDocException – SAXExceptionist tuletatud JDigiDoc exception

- ee.sk.digidoc.factory.NotaryFactory – Kehtivuskinnituse hankimist kirjeldav interface
- ee.sk.digidoc.factory.IAIKNotaryFactory - Kehtivuskinnituse hankimine IAIK JCE teegi abil. Vajalik IAIK JCE teegi litsensi hankimine!
- ee.sk.digidoc.factory.BouncyCastleNotaryFactory - Kehtivuskinnituse hankimine BouncyCastle teegi abil. See variant baseerub ainult vabavarale!
- ee.sk.digidoc.factory.CanonicalizationFactory – XML kanoniseerimist kirjeldav interface
- ee.sk.digidoc.factory.DOMCanonicalizationFactory – XML kanoniseerimine Apache XML Security teegi abil.
- ee.sk.digidoc.factory.SigantureFactory – Allkirjastamist kirjeldav interface
- ee.sk.digidoc.factory.PKCS11SigantureFactory – Allkirjastamine PKCS#11 liidese abil.

Pakett: ee.sk.util

Selles paketis on konfiguratsiooni ja utility klassid.



- ee.sk.util.ConfigManager – Konfiguratsiooni andmete lugeja
- ee.sk.util.ConvertUtils – Konverteerimismeetodeid sisaldav utility klass.
- ee.sk.util.LabelsManager – Kasutajaliideses kasutatud tekstide eri keelsete versioonide lugeja. Hetkel ei kasutata.

Pakett: ee.sk.test

Selles paketis on näiteprogrammid

- ee.sk.test.Test1 – loob DIGIDOC-XML 1.3 formaadis faili, lisab sinna kolm andmefaili (üks igast lubatud formaadist), allkirjastab ID kaardi ja PKCS#11 abil, hangib kehtivuskinnituse kirjutab faili, loeb uuesti failist ja kontrollib allkirju ja kehtivuskinnitusi.
- ee.sk.test.Test2 – Loeb DIGIDOC-XML 1.3 formaadis faili ja kontrollib allkirju ning kehtivuskinnitusi.

Põhilised toimingud

Järgnevas kirjeldame põhiliste toimingute kasutamist JDigiDoc teegis.

Initsialiseerimine

JDigiDoc teegi initsialiseerimiseks loeme sisse konfiguratsiooniandmed:

```
ConfigManager.init("jar://JDigiDoc.cfg");
```

Failide loomine

Uue DIGIDOC-XML formaadis faili loomiseks tekitame uue SignedDoc objekti:

```
SignedDoc sdoc = new SignedDoc(SignedDoc.FORMAT_DIGIDOC_XML,
SignedDoc.VERSION_1_3);
```
Võimalik on kasutada ka versiooni 1.2 või 1.1 (`SignedDoc.VERSION_1_2`, `SignedDoc.VERSION_1_1`). See objekt tekitatid mälus ja teda ei ole veel faili kirjutatud.

Andmefailide lisamine

Andmefailid võivad olla EMBEDDED_BASE64 kujul (binary failid), EMBEDDED kujul (puhas XML või tekst) ja DEATCHED kujul (viide välisele failile).

- EMBEDDED_BASE64 kujul faili lisamiseks teeme:

```
DataFile df = sdoc.addDataFile(new File(<faili asukoht>),
          <mime tüüp>, DataFile.CONTENT_EMBEDDED_BASE64);
```
- EMBEDDED kujul faili lisamiseks teeme:

```
DataFile df = sdoc.addDataFile(new File(<faili asukoht>),
          <mime tüüp>, DataFile.CONTENT_EMBEDDED);
```
- DETATCHED kujul faili lisamiseks teeme:

```
DataFile df = sdoc.addDataFile(new File(<faili asukoht>),
          <mime tüüp>, DataFile.CONTENT_DETATCHED);
```
Lisatud faili jaoks tekitatakse objekt mälus aga andmefaili veel ei loeta. Andmefaili loetakse alles DIGIDOC faili kirjutamise momendil. Andmefaile loetakse ka allkirjastamise momendil, kuna allkirjastamiseks on vaja arvutada nende räsid.

Kui te ei soovi, et JDigiDoc andmefaile loeb, sest te hoiate neid andmeid baasis, genereerite dünaamiliselt vms., siis omistage andmefaili sisu ise DataFile objektile. Kui DataFile objektile on sisu omistatud siis hoitakse seda mälus ja ei minda enam andmefaili kettalt lugema. Selleks kasutame meetodit setBody():

```
String myBody = "Minu andmed koos täpitähtedega";
df.setBody(myBody.getBytes("ISO-8859-1"), "ISO-8859-1");
```
Antud juhul eeldame et algandmed on kliendi poolt kasutatavas tähestikus, näiteks ISO-8859-1. Sel juhul genereerib JDigiDoc XML faili elemendile DataFile atribuudi Codepage="ISO-8859-1" mida kasutatakse failist andmete lugemisel et andmeid algsesse tähestikku tagasi konverteerida. DigiDoc failis hoitakse andmeid UTF-8 kujul.

Kui teie andmed juba on UTF-8 -s, siis teeme:

```
byte[] u8b = ConvertUtils.str2data(myBody);
df.setBody(u8b, "UTF-8");
```
Kui andmefail koosnes puhtast XML-st siis saab seda lugeda meetodiga:

```
String body = df.getBodyAsString();
```

Allkirjastamine

Allkirjastamiseks tuleb kasutada SignatureFactory interfacet. ID kaardi jaoks saab kasutada PKCS#11 ohjurprogrammi või mingit muud (CSP?) ohjurprogrammi kasutavat välist programmi. Esmalt tuleb hankida kasutaja sertifikaat. Selleks teeme PKCS#11 ohjurprogrammi kasutamisel:

```
String pin = "<ID kaardi PIN2>";
SignatureFactory sigFac = ConfigManager.
                instance().getSignatureFactory();
# ID kaardil on vaid 1 digiallkirjsertifikaat, seega index 0
X509Certificate cert = sigFac.getCertificate(0, pin);
```
Nüüd lisame allkirja andmed ja arvutame allkirja räsikoodi:

```
Signature sig = sdoc.prepareSignature(cert,
                    null, // String[] claimedRoles,
                    null); // SignatureProductionPlace address
byte[] sidigest = sig.calculateSignedInfoDigest();
```
Allkiri ei ole veel valmis, kuna puudub tegelik allkirja väärtus. Selle arvutamiseks võib

kasutada ka välist programmi, mis antud 20 baidise SHA1 räsikoodi ID kaardiga allkirjastab. PKCS#11 ohjurprogrammi kasutamiseks teeme nii:

```
byte[] sigval = sigFac.sign(sidigest, 0, pin);
```

Siis lisame allkirja väärtuse allkirja objektile:

```
sig.setSignatureValue(sigval);
```

Kehtivuskinnituse hankkimine

Kehtivuskinnituse hankkimiseks tuleb kasutada NotaryFactory interfacet, kuid seda teeb allkirja objekt ise:

```
sig.getConfirmation();
```

Peale kehtivuskinnituse hankkimist on allkiri lõplikult valmis ja omab pikaajalist tõendusväärtust.

Kui soovite kasutada antud teeki kehtivuskinnituste hankimiseks mingist muust rakendusest aga teil ei ole tegu DigiDoc formaadis failidega või soovite seda implementeerida ise ning kasutada ainult selle teegi kehtivuskinnituste osa funktsionaalsust, siis kasutage meetodit:

```
NotaryFactory notFac = ConfigManager.
                instance().getNotaryFactory();
Notary not = notFac.getConfirmation(byte[] nonce,
        X509Certificate signersCert, String notId)
        throws DigiDocException;
```

Kui kehtivuskinnituste ei õnnestunud hankida, siis tekib viga.

Puhtalt sertifikaadi enda kehtivuse kontrolliks kasutage meetodit:

```
public void NotaryFactory.checkCertificate(X509Certificate cert)
        throws DigiDocException;
```

Failide kirjutamine ja lugemine

Loodud SignedDoc kirjutame fali järgmiselt:

```
sdoc.writeToFile(new File("<faili-asukoht-ja-nimi>"));
```

Kui te ei soovi kirjutada faili vaid mingile muule andmekandjale, andmebaasi vms. siis kasutage meetodit: SignedDoc.writeToStream(OutptStream os).

Allkirjade ja kehtivuskinnituste kontroll

Olles lugenud sisse DigiDoc faili loetleme kõik allkirjad ja kontrollime neid:

```
ArrayList errs = sdoc.verify(true, true);
if(errs.size() == 0)
  System.out.println("OK");
for(int j = 0; j < errs.size(); j++)
  System.out.println((DigiDocException)errs.get(j));
```

Antud meetod kontrollib antud dokumendi iga allkirja kehtivust. Juhul kui allkirjal on kehtivuskinnitus siis kontrollitakse ka seda. Meetodi verify() esimene parameeter on tõeväärtus mis määrab, kas kontrollida allkirjastajate sertide kehtivust (serdi kehtivuse algus- ja lõppkuupäeva järgi). Kui see on false siis sellist kontrolli ei tehta. Selline kontroll on vähem täpne kui kontroll OCSP kehtivuskinnituse järgi, mille teostamist saab nõuda teise parameetri abil. Esimene kontroll on vajalik vaid siis kui te ei ole hankinud oma allkirjale kehtivuskinnitust (näiteks kui te ei kasuta Eesti ID kaarti allkirjastamiseks).

Teine parameeter on lipp (boolean), mis näitab kas nõuda igalt allkirjalt kehtivuskinnitust või mitte. Kui kehtivuskinnitus on nõutud ja allkirjal ei ole kehtivuskinnitust, siis tekib viga ja faili ei loeta korrektselt allkirjastatuks.

Vigade leidmisel ei visata Exception objekti vaid salvestatakse kõik Exception obejktid ArrayList konteineris. See võimaldab mitme vea puhul kõik vead leida.

Erinevused JDigiDoc versioonide 1.1 ja eelnevate vahel

- Logimine – JDigiDoc kasutab nüüd Log4j teeki logimiseks. Seoses sellega on ka lisandunud DIGIDOC_LOG4J_CONFIG, mis näitab Log4j config faili asukohta. On võimalik salvestada ka saadetud OCSP päringud ja saadud vastused. Selleks defineeri ka `OCSP_SAVE_DIR`
- CRL kasutamine - JDigiDoc toetab ka CRL kasutamist sertifikaadi kehtivuse kontrolliks. Selleks tuleb defineerida:
    - CRL_FACTORY_IMPL=<CRL implementatsiooniklass>
    - DIGIDOC_CERT_VERIFIER=<serdi kontrollija: OCSP või CRL>
    - DIGIDOC_SIGNATURE_VERIFIER=<allkirja kontrollija: OCSP või CRL>
    - CRL_USE_LDAP=false
    - CRL_FILE=<fail kuhu kirjutada ajutiselt CRL>
    - CRL_URL=http://www.sk.ee/crls/esteid/esteid.crl
    - CRL_SEARCH_BASE=cn=ESTEID-SK,ou=ESTEID,o=AS Sertifitseerimiskeskus,c=EE
    - CRL_FILTER=(certificaterevocationlist;binary=*)
    - CLR_LDAP_DRIVER=com.ibm.jndi.LDAPCtxFactory
    - CRL_LDAP_URL=ldap://194.126.99.76:389
    - CRL_LDAP_ATTR=certificaterevocationlist;binary
    - CRL_PROXY_HOST=<teie proxy host>
    - CRL_PROXY_PORT=<teie proxy port>
- AID kaartide toetus - JDigiDoc toetab ka AID kaartidega allkirjastamist ja nende allkirjade kontrolli. Allkirjastamaks AID kaardiga saab kasutada:
    - DIGIDOC_SIGN_PKCS11_DRIVER=pk2priv
- Sertifikaadi kontroll CA abil - JDigiDoc võib kontrollida kas allkirjastaja sertifikaat on väljastatus mõne usaldatud CA poolt. Selline kontroll eelneb OCSP kontrollile ja ei ole nõutud. Kui CA sertide arv defineerida 0 siis seda kontrolli ei tehta. Vastasel juhul seisneb kontroll selles, et allkirjastaja sertifikaat peab olema ühe loetletud CA serdi poolt otseselt väljaantud. Kui soovitakse kasutada siis tuleb defineerida:
    - DIGIDOC_CA_CERTS=<CA sertide arv>
    - DIGIDOC_CA_CERT<n>=<mingi CA serdi PEM fail>. Seejuures N algab 1-st.
- Mitme OCSP responderi tugi – OCSP responderi aadress on ikka http://ocsp.sk.ee aga kui allkirjastate AID või mingi test- või demokaardiga (FinID, etc.) siis vastab teine responder ja tollel on ka erinev responderi ID ning CA sertide hierarhia. JDigiDoc toetab mitme responderi kasutamist saates esmalt päringu teele ja kontrollides vastuse saamisel milline responder vastas ning valides automaatselt sellele responderile sobivad sertifikaadid. Muutnud on ka NotaryFactory meetodid getNotaryCert() ja getCACert() kus nüüd tuleb edastada ka soovitud responderi CN sest neid on ju mitu. Kui soovitakse kasutada siis tuleb defineerida:
    - DIGIDOC_OCSP_COUNT=<tuntud responderite arv sertide arv>
    - DIGIDOC_OCSP<n>_CN=<antud responderi common name>
    - DIGIDOC_OCSP<n>_CERT=<antud responderi sertifikaat>

    - DIGIDOC_OCSP<n>_CA_CERT=<antud responderi CA sertifikaat>
    - DIGIDOC_OCSP<n>_CA_CN=<antud responderi CA common name>
- Embedded Bas64 hoidmine mälus – Kui ennem hoidis teek embedded base64 andmetüübi puhul dekodeeritud andmeid, siis nüüd hoitakse kogu base64 <DataFile> sisu. Seda selleks et vältida allkirja valeks muutumist kui <DataFile> sisaldab mingeid reavahetusi või whitespace sümboleid base64 andmete ees, järel või sees.
- Formaat 1.3 – JDigiDoc toetab nüüd formaati 1.3 ning endiselt formaate 1.1 ja 1.2. Formaadi 1.3 erinevused on dokumenteeritud DigiDoc formaadi spetsifikatsioonis.
- Allkirjade cachimine – JDigiDoc hoiab nüüd failist loetud allkirju mälus et vältida nende valeks minemist kui uus allkiri lisada ja uuesti salvestada. See tagab ka parema ühilduvuse CdigiDoc-ga
- Dokumendid kasutavad nüüd SignedDoc namespacet.
- Lisatud on SignedDoc.removeDataFile() meetod.
- DataFile klassi kontruktorile lisati parent objekti - SignedDoc viide.
- DataFile atribuut Filename tohib nüüd sisaldada ka '&' sümbolit.
- Loodi uus BouncyCastleNotaryFactory mille abil saab samuti hankida kehtivuskinnitusi ja mis kasutab vabavara Bouncy-Castle Java cryptoteeki. Leiti, et tuleb kasutada uusimat 1.23 versiooni, kuna 1.22-s oli viga.
- SignedDoc.verify() esimene parameeter oli java.lang.Date ja määras allkirjastaja sertifikaadi kontrollimise kuupäeva. Kuna aga allkirjad võivad olla antud erinevatel aegadel siis see ei toiminud korrektselt. Nüüd on see boolean parameeter mis määrab kas sellist kontrolli teha ja kui nii siis kasutatakse allkirjastamise kuupäeva sertifikaadi kehtivuse kontrolliks.
- Ennem ei toiminud teek juhul kui allkirjastaja sertifikaat ja kehtivuskinnituse sertifikaat olid väljaantud erinevate CA -de poolt. Nüüd on see sõltuvus kaotatud.