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.

Lisaks digiallkirjastamisele pakub JDigiDoc ka krüpteerimist ja dekrüpteerimist vastavalt XML-ENC standardile. Antud standardi käsitleb XML dokumentide või nende osade krüpteerimist aga lubab krüpteerida ka suvalisi binaarfaile kodeerides nad eelenevalt Base64 kodeeringus. JDigiDoc pakub ka võimalust krüpteeritavaid andmeid eelnevalt komprimeerida ZLIB algoritmi abil. Krüpteerimine toimib 128 bitise AES transpordivõtme abil, mis omakorda krüpteeritakse vastuvõtja(te) sertifikaadiga.

JDigiDoc klassid järgivad küllaltki täpselt XML-DSIG, ETSI ja XML-ENC 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
- XML-ENC

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.BouncyCastleNotaryFactory
DIGIDOC_FACTORY_IMPL=ee.sk.digidoc.factory.SAXDigiDocFactory
CANONICALIZATION_FACTORY_IMPL=ee.sk.digidoc.factory.DOMCanonicalizationFactory
CRL_FACTORY_IMPL=ee.sk.digidoc.factory.CRLCheckerFactory
ENCRYPTED_DATA_PARSER_IMPL=ee.sk.xmlenc.factory.EncryptedDataSAXParser
ENCRYPTED_STREAM_PARSER_IMPL=ee.sk.xmlenc.factory.EncryptedStreamSAXParser
```

- Java krüptoteekide valik.

```
# Security settings
DIGIDOC_SECURITY_PROVIDER=org.bouncycastle.jce.provider.BouncyCastleProvider
DIGIDOC_SECURITY_PROVIDER_NAME=BC
```

- 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=<kataloog-teie-systeemis>\\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>
```

- Krüpteerimise seaded

```
DIGDOC_ENCRYPT_KEY_ALG=AES
DIGIDOC_ENCRYPTION_ALOGORITHM=AES/CBC/PKCS7Padding
DIGIDOC_SECRANDOM_ALGORITHM=SHA1PRNG
DIGIDOC_KEY_ALOGORITHM=RSA/NONE/PKCS1Padding
```

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
log4j.logger.ee.sk.xmlenc.factory.EncryptedDataSAXParser=INFO, output
log4j.logger.ee.sk.xmlenc.factory.EncryptedStreamSAXParser=INFO, output
log4j.logger.ee.sk.digidoc.DataFile=INFO, output
log4j.logger.ee.sk.xmlenc.EncryptedData=INFO, output
log4j.logger.ee.sk.xmlenc.EncryptedKey=INFO, output
log4j.logger.ee.sk.digidoc.Base64Util=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
```

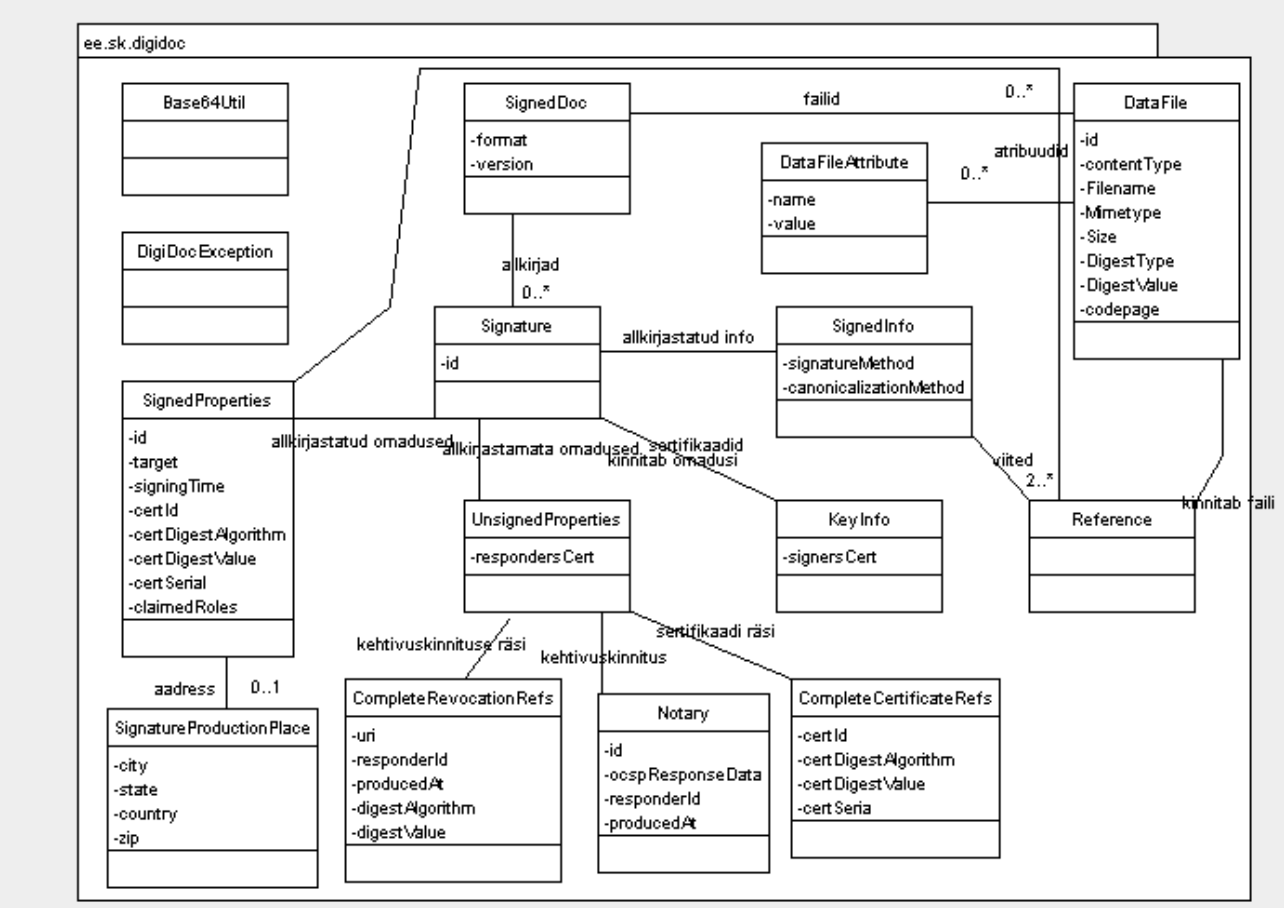
JDigiDoc arhitektuurist

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
- ee.sk.xmlenc – JDigiDoc krüpteerimise klassid vastavalt XML-ENC standardi järgi defineeritud XML elementidele
- ee.sk.xmlenc.factory – Sisaldab krüpteerimisel kasutatavaid mooduleid mida saab vahetada omaloodute vastu. Hetkel sisaldab kahte parser moodulit millest üks on üldine krüpteeritud dokumendi parsija ja sobib vaid väikeste dokumentide jaoks ja teine on kavandatud suurte krüpteeritud dokumentide dekrüpteerimiseks.

Pakett: ee.sk.digidoc

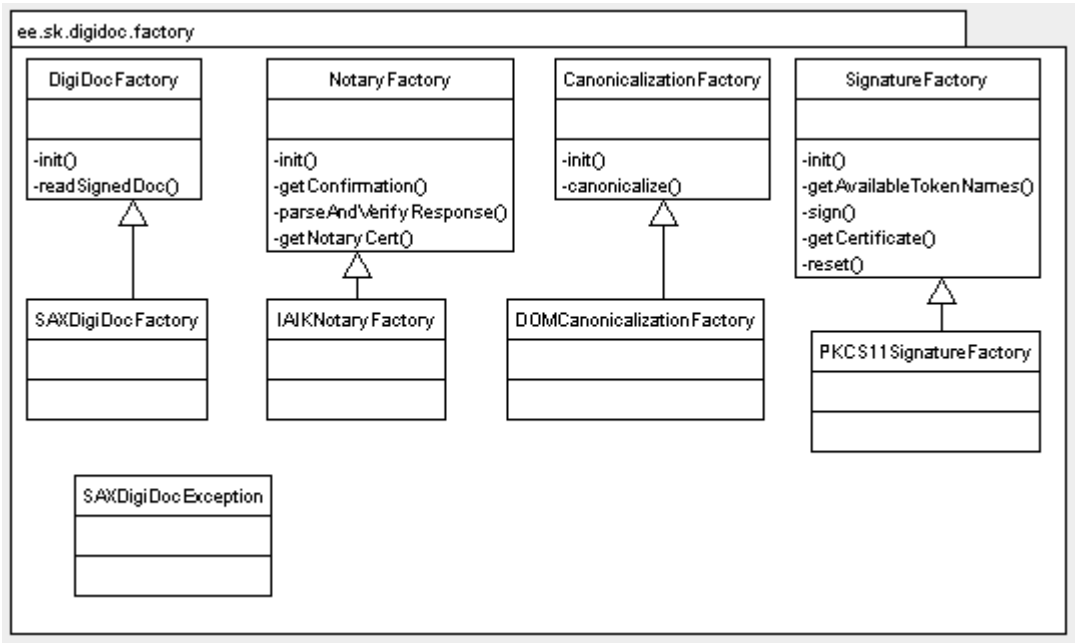
Selles paketis on põhilised JDigiDoc klassid, mis järgivad DigiDoc failide XML elementide struktuuri.



- 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

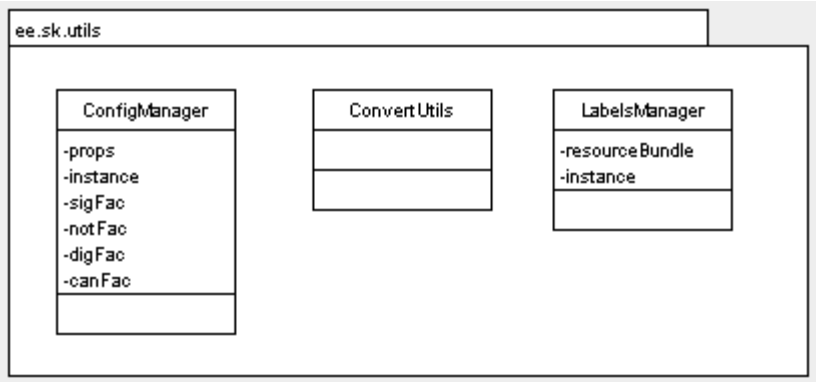
Selles paketis on mitmesuguseid algoritme implementeerivad klassid ja antud funktsionaalsust defineerivad intefacesid.



- 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.

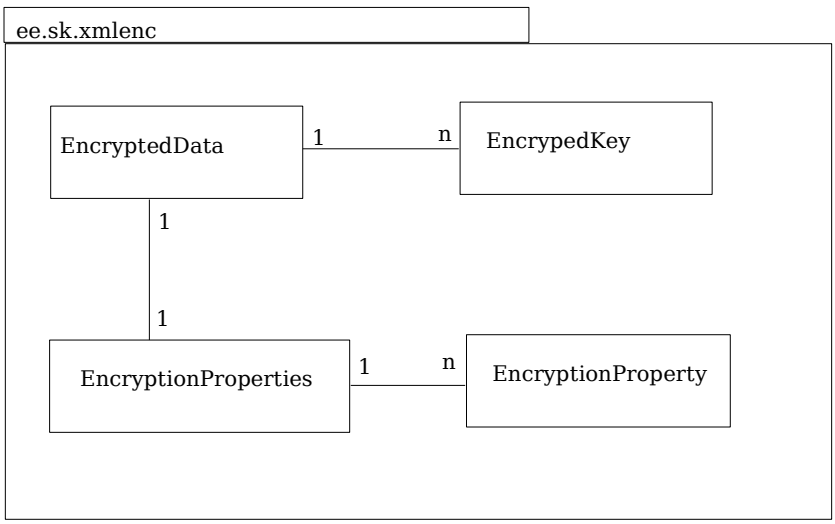
Pakett: ee.sk.test

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.
- ee.sk.test.jdigidoc – universaalne demoprogramm mis pakub käsurealt erinevate teegi võimaluste kasutamist.

Pakett ee.sk.xmlenc

Selles paketis on XML-ENC standardile vastavate XML elementide klassid.



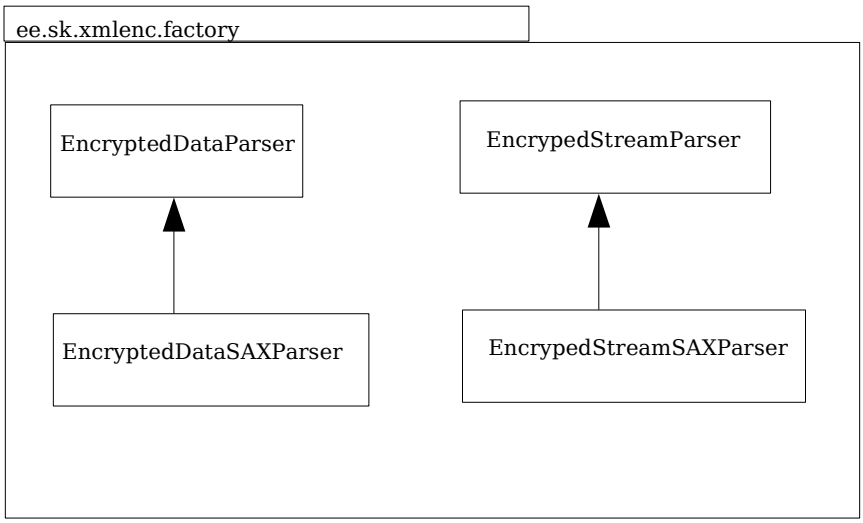
- ee.sk.xmlenc.EncryptedData – Vastab XML-ENC standardi <EncryptedData>

elemendile. JDigiDoc ei toeta mitut krüpteeritud andmekogust ühes failis või krüpteeritud ja krüpteerimata andmete segu. Ühes krüpteeritud failis on vaid üks <EncryptedData> element, mis sisaldab ühe <EncryptedKey> elemendi iga võimaliku vastuvõtja/dekrüpteerija jaoks ja vastavalt vajadusele <EncryptionProperty> elemente milles salvestatakse

- ee.sk.xmlenc.EncryptedKey - Vastab XML-ENC standardi <EncryptedKey> elemendile. Ühes krüpteeritud dokumendis on iga võimaliku vastuvõtja jaoks üks selline <EncryptedKey> element mis sisaldab selle vastuvõtja ID kaardi autentimissertifikaadiga krüpteeritud AES transpordivõtit.
- ee.sk.xmlenc.EncryptionProperties - Vastab XML-ENC standardi <EncryptionProperties> elemendile. Sisaldab ühte või enamat EncryptionPropety objekti.
- ee.sk.xmlenc.EncryptionProperty - Vastab XML-ENC standardi <EncryptionProperty> elemendile. XML-ENC standard defineerib lubatud (kuid mitte nõutud) atribuudid “Id” ja “Target” ning lubab ka muid atribuute lisada. JDigiDoc lisab atribuudi “Name” mille järgi eristatakse spetsiaalseid elemente mida teek kasutab krüpteeritud obejtki teatud omaduste salvestamiseks krüpteeritud dokumendis.

Pakett ee.sk.xmlenc.factory

Selles paketis on krüpteeritud failide parserid.



- ee.sk.xmlenc.factory.EncryptedDataParser – defineerib krüpteeritud dokumendi üldise parseri. Antud parser sobib väksemate dokumentide lugemiseks.
- ee.sk.xmlenc.factory.EncryptedDataSAXParser – implementeerib EncryptedDataParser interfacet SAX parseri abil.
- ee.sk.xmlenc.factory.EncryptedStreamParser – defineerib suurte krüpteeritud dokumentide parseri. Antud parser sobib suuremate dokumentide dekrüpteerimiseks.
- ee.sk.xmlenc.factory.EncryptedStreamSAXParser- implementeerib EncryptedStreamParser interfacet SAX parseri abil.

Digiallkirjastamine

JDigiDoc teek pakub digiallkirjastatud dokumentide koostamist, allkirjastamist, allkirjade kontrolli ja lugemist vastavalt XAdES (ETSI TS101903) ja XML-DSIG standarditele. Järgnevates peatükkides kirjeldatakse digiallkirjastase põhilisi operatsioone JDigiDoc teegiga.

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.

Krüpteerimine ja dekrüpteerimne

Lisaks digiallkirjastamisele pakub JDigiDoc ka krüpteerimist ja dekrüpteerimist vastavalt XML-ENC standardile. Antud standardi käsitleb XML dokumentide või nende osade krüpteerimist aga lubab krüpteerida ka suvalisi binaarfaile kodeerides nad eelenevalt Base64 kodeeringus. JDigiDoc pakub ka võimalust krüpteeritavaid andmeid eelnevalt komprimeerida ZLIB algoritmi abil. Krüpteerimine toimib 128 bitise AES transpordivõtme abil, mis omakorda krüpteeritakse vastuvõtja(te) sertifikaadiga. Järgnevates peatükkides kirjeldatakse põhilisi krüpteerimis- ja dekrüpteerimisoperatsioone JDigiDoc teegiga.

Krüpteeritud dokumentide koostamine

Krüpteeritud dokumendi koostamiseks tuleb esmalt luua EncryptedData objekt millele siis järgenvas lisatakse krüpteeritavad andmed ja transpordivõtmed ning siis kirjutatakse faili.

```
EncryptedData cdoc = new EncryptedData(
          null, // mitte nõutud Id atribuudi väärtus
          null, // mitte nõutud Type atribuudi väärtus
          null, // mitte nõutud MimeType atribuudi väärtus
          EncryptedData.DENC_XMLNS_XMLENC,  // fikseeritud xml namespace
          EncryptedData.DENC_ENC_METHOD_AES128); // fikseeritud krüptoalgoritm
```

Mitte nõutud atribuutide väärtutstena tuleb kindlasti edastada null mitte näiteks tühi string kui ei soovita neid kasutada. Kui krüpteeritakse digidoc dokumenti, siis tuleks atribuudile “Type” omistada väärtus: "http://www.sk.ee/DigiDoc/v1.3.0/digidoc.xsd" mis on defineeritud ka konstandina EncryptedData.DENC_ENCDATA_TYPE_DDOC. Kui krüpteeritakse XML dokuemente siis võiks atribuudile “MimeType” omistada väärtuse “text/xml” mis on defneeritud ka konstandina EncryptedData.DENC_ENCDATA_MIME_XML. JDigiDoc teek kasutab atribuuti “MimeType” ka selleks et märkida et krüpteeritud andmed on eelnevalt ka pakitud ZLIB algoritmi abil. Siis omistatakse atribuudile väärtus "http://www.isi.edu/in-noes/iana/assignments/media-types/application/zip", mis on defineeritud ka konstandina EncryptedData.DENC_ENCDATA_MIME_ZLIB. Seda ei ole vaja ise teha. Teek omistab selle väärtuse automaatselt andmeid pakkides. Kui enne seda oli atribuudile MimeType omistatud mingi muu väärtus, siis salvestatakse see alamelementi <EncryptionProperty Name=”OriginalMimeType”>. Kui teek märkab dokumendi dekrüpteerimisel antud atribuuti siis üritatakse andmeid peale dekrüpteerimist ka dekomprimerida ja pealse seda taastatakse algne MimeType kui see eksisteerib.

Dokumendi vastuvõtjate registreerimine

Igal krüpteeritud dokumendil peaks olema vähemalt üks või mitu “vastuvõtjat” ehk isikut kes suudavad antud dokumenti dekrüpteerida. Iga vastuvõtja jaoks krüpteeritakse andmete krüpteerimiseks kasutatud AES transpordivõti antud isiku sertifikaadiga. Kasutada tuleb kindlasti sellist sertifikaati mis sobib andmete krüpteerimiseks ehk ID kaardi puhul isiku autentimissertifikaati. EncryptedKey objekti lisamiseks tuleb omada isiku sertifikaati PEM formaadis ja lisanduvalt võib loodavale objektile omistada ja “Id” ja “Recipient” atribuute ning <KeyName> ja/või <CerriedKeyName> alamelemente. Kõik nimetatud andmed ei ole nõutud aga neid võib kasutada soovitud vastuvõtja transpordivõtme otsimiseks muude hulgast või näiteks andtud transpordivõtme kuvamiseks ekraanil. EncryptedKey objekti lisame järgmiselt:

```
X509Certificate recvCert = SignedDoc.readCertificate(new File(certFile));
EncryptedKey ekey = new EncryptedKey(
          null,     // mitte nõutud Id atribuudi väärtus
          null,     // mitte nõutud Recipient atribuudi väärtus
          EncryptedData.DENC_ENC_METHOD_RSA1_5, // fikseeritud krüptoalgoritm
          null,     // mitte nõutud KeyName alamelemendi väärtus
          null,     // mitte nõutud CarriedKeyName alamelemendi väärtus
          recvCert); // vastuvõtja sertifikaat. Nõutud!

cdoc.addEncryptedKey(ekey);
```

Näiteprogramm ee.sk.test.jdigidoc omistab igale EncryptedKey objektile mingi nime kasutades selleks “Recipient” atribuuti. See võiks olla näiteks vastuvõtja eesnimi või midagi keerukamat nagu näiteks “<perekonnanimi>,<eesnimi>,<isikukood>”. Hiljem saab käsurealt sisestada dekrüpteerimise operatsioonil antud nime et identifitseerida soovitud vastuvõtjat (ja seega <EncryptedKey> objekti) kelle ID kaarti kasutatakse dekrüpteerimiseks. Kuivõrd ainsaks nõutud parameetriks on vastuvõtja sertifikaat siis oleks mõistlik baseerida rakenduse loogika EncryptedKey valikuks mingile sertifikaadi parameetrile nagu näiteks sertifikaadi CN atribuudile.

Krüpteerimine ja salvestamine

Krüpteerimiseks on kaks eri meetodit:

- väikesete objektide jaoks – hoiab kõiki andmeid mälus. Kiirem ja paindlikum aga nõuab rohkem mälu. Võimalik on pakkimisel kasutada valikut “BEST EFFORT” mis tähendab et andmed komprimeeritakse ja kui tulemuseks oli andmete mahu vähenemine siis neid kasutatakse, vastasel juhul kasutatakse andmeid algkujul.
- Suurte andmehulkade jaoks – loeb, töötleb ja kirjutab andmeid blokkhaaval. Pisut aeglasem aga suudab krüpteerida suvalise suurusega objekte. Pakkimisel tuleb kindlalt määrata kas pakkimist teostada või mitte.

Väikeste andmehulkade krüpteerimiseks tuleb esmalt luua EncryptedData objekt, siis lisada sellele kõik vastuvõtjate andmed, siis lugeda sisse ja omistada krüpteeritavad andmed ja lõpuks krüpteerida ning kirjutada faili või kasutada muul kombel loodud krüpteeritud xml dokumenti. Näiteks:

```
// loeme sisse krüpteerimata andmed
byte[] inData = SignedDoc.readFile(new File(inFile));
cdoc.setData(inData);
cdoc.setDataStatus(EncryptedData.
```

```
DENC_DATA_STATUS_UNENCRYPTED_AND_NOT_COMPRESSED);
// salvestame hilismaks algse failinime kui andmed olid failis
cdoc.addProperty(EncryptedData.ENCPROP_FILENAME, inFile);
// Komprimerime. Valikud: EncryptedData.DENC_COMPRESS_ALLWAYS,
// EncryptedData.DENC_COMPRESS_NEVER ja EncryptedData.DENC_COMPRESS_BEST_EFFORT
cdoc.encrypt(EncryptedData.DENC_COMPRESS_BEST_EFFORT);
FileOutputStream fos = new FileOutputStream(outFile);
fos.write(m_cdoc.toXML());
fos.close();
```

Suurte andmehulkade komprimeerimiseks tuleb alguses samuti koostada EncryptedData objekt ja registreerida kõigi vastuvõtjate andmed ja seejärel teeme:

```
// salvestame algse sisendfaili nime kui vajalik
cdoc.addProperty(EncryptedData.ENCPROP_FILENAME, inFile);
// Krüpteerime. Valikud on ainult EncryptedData.DENC_COMPRESS_ALLWAYS
// ja EncryptedData.DENC_COMPRESS_NEVER
cdoc.encryptStream(new FileInputStream(inFile),
          new FileOutputStream(outFile), EncryptedData.DENC_COMPRESS_ALLWAYS);
```

Nii ühel kui teisel juhul ei pea tulemust mitte ilmtingimata faili kirjutama vaid võib ka näiteks baidijadasse kirjutada ja emailiga saata vms.

Dekrüpteerimine ja failide lugemine

Krüpteeritud dokumentide lugemiseks on kaks eri liidest:

- EncryptedDataParser – sobib väiksemate krüpteeritud dokumentide lugemiseks. Peale lugemist on andmed mälus. Neid saab saab seal dekrüpteerida või lihtsalt kuvada. Parser ei teosta automaatselt mingit dekrüpteerimisoperatsiooni.
- EncryptedStreamParser – sobib suuremate krüpteeritud dokumendtide dekrüpteerimiseks. Parserile tuleb edastada kõik dekrüpteerimiseks vajalik ja tulemuseks ei koguta andmeid mällu vaid kirjutatakse kohe faili. Vaja on teada vastuvõtja mingit atribuuti mille abil valitakse välja dekrüpteerimiseks kasutava EncryptedKey objekt.

Väikeste krüpteeritud dokumentide lugemiseks teeme:

```
EncryptedDataParser dencFac =  ConfigManager.instance().
          getEncryptedDataParser();
cdoc = dencFac.readEncryptedData(inFile);
```

Nüüd on andmed krüpteeritud ja võibolla ka pakitud kujul mälus. Võimalik on kasutada EncryptedData, EncryptedKey ja EncryptionProperty objektide meetodeid andmete kuvamiseks või andmed dekrüpteerida järgmselt:

```
m_cdoc.decrypt(0,  // EncryptedKey objekti järjekorra number
          0,      // kiipkaardi Tokeni number. Eesti ID kaardil alati 0
          pin);   // kiipkaardi PIN kood. Eest ID kaardil PIN1
FileOutputStream fos = new FileOutputStream(outFile);
fos.write(m_cdoc.getData());
fos.close();
```

Suure andmehulkade dekrüpteerimiseks teeme:

-

```
// avame sisend- ja väljundjadad
FileInputStream fis = new FileInputStream(inFile);
FileOutputStream fos = new FileOutputStream(outFile);
```

```
EncryptedStreamParser streamParser = ConfigManager.
          instance().getEncryptedStreamParser();
// dekrüpteerime
streamParser.decryptStreamUsingRecipientName(fis, fos,
          0,        // kiipkaardi Tokeni number. Eesti ID kaardil alati 0
          pin,      // kiipkaardi PIN kood. Eest ID kaardil PIN1
          recvName); // vastuvõtja EncryptedKey atribuudi Recipient väärtus
fos.close();
fis.close();
```

Andmed loetakse sisendjadast, dekrüpteeritakse, kui vaja siis dekomprimeeritakse ja kirjutatakse otse väljundjadasse.

Valideerimine

Mõlemad “dokumendi tasemel” objektid – SignedDoc ja EncryptedData omavad meetodeid validate() mis tagastab ArrayList loetely DigiDocException objektidest – mis sisaldavad formaadi vigasid. Hoolimata formaadi vigadest võib dokumendi allkirju kehtivaks lugeda või suuta antud dokumenti dekrüpteerida kuid programm mis lisab dokumendile allkirja või krüpteerib andmeid peaks kindlasti peale sellist toiminud kontrollima saadud dokumendi formaati et olla veendunud et teegi võimalisi on õieti kasutatud.

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 atribuuti 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.

Erinevused JDigiDoc versioonide 2.0 ja 1.1 vahel

- JDigiDoc toetab nüüd ka XML-ENC standardi järgi krüpteerimist ja dekrüpteerimist.
- Nõutud on veel järgmised konfiguratsioonifaili kirjed:

```
ENCRYPTED_DATA_PARSER_IMPL=ee.sk.xmlenc.factory.EncryptedDataSAXParser
ENCRYPTED_STREAM_PARSER_IMPL=ee.sk.xmlenc.factory.EncryptedStreamSAXParser
DIGIDOC_SECURITY_PROVIDER_NAME=BC
DIGDOC_ENCRYPT_KEY_ALG=AES
DIGIDOC_ENCRYPTION_ALOGORITHM=AES/CBC/PKCS7Padding
DIGIDOC_SECRANDOM_ALGORITHM=SHA1PRNG
DIGIDOC_KEY_ALOGORITHM=RSA/NONE/PKCS1Padding
```

- Enam ei ole vaja IAIKBouncyCastle ja IAIK security provideri kirjeid kuna selle asemel saab kasutada ka BouncyCastle teegile baseeruvaid OCSP teenuseid. Järgmised kirjed ei ole enam vajalikud

```
IAIK_SECURITY_PROVIDER=iaik.security.provider.IAIK
DIGIDOC_NOTARY_IMPL=ee.sk.digidoc.factory.IAIKNotaryFactory
```

- Loodud on ka käsureautiliit – ee.sk.test.jdigidoc, mille abil saab teostada enamikku teegi poolt võimalikke teenuseid.

Erinevused JDigiDoc versioonide 2.1.1 ja 2.1.0 vahel

- Klassi DataFile parameetriteta konstruktor eemaldati. See konstruktor lasi tekkida programmidel mis genereerisid ebakorrektse vorminguga digidoc dokumente ja seetõttu eemaldati see meetod. Objekti DataFile atribuudile “id” peab omistama unikaalse tunnuse vormis “D<number>” ja parim variant sellise genererimiseks oleks kasutada objekti SignedDoc meetodit getNewDataFileId().
- Lisati meetod X509Certificate SignedDoc.readCertificate(String certLocation); mis lisaks sertifikaadi lugemisele failist võib lugeda sertifikaati ka URL -lt ja kui kasutada URL-i stiilis jar://<location> siis ka suvalisest kohast mis asub CLASSPATH-s.
- Objekti ConfigManager meetodit init() muudeti nii et ta ei eemaldaks ennem mälus olnud konfiguratsioonikirjeid vaid ainult lisaks uusi. Konfiguratsioonikirjete eemaldamiseks lisati meetod ConfigManager.reset(). Seega on nüüd võimalik kombineerida mitme konfiguratsioonifaili või muu konfiguratsioonikirjete allika sisu. Koos eelmise muudatusega tähendab see et on võimalik tekitada standardtne konfiguratsioonifail mis salvestatakse teegi (JDigiDoc-x.y.jar) sisse ja ka enamus standardsetest CA sertifikaatidest võivad asuda selles .jar failis. Kasutaja peab looma vaid väiksema konfiguratsioonifaili mis sisaldab tema poolt lisatud erinevaid kirjeid (PKCS12 juurdepääsutõend, HTTP proxy jms.)
- Klassidele EncryptedData, EncryptedKey, EncryptionProperty ja EncryptionProperties lisati validate() meetod mis tagastab ArrayList objekti koos tuvastatud formaadi vigadega.
- Meetoditele SignedDoc.validate() ja DataFile.validate() lisati parameeter **boolean bStrong** mis määrab kas kontrollida ka Id atribuudi sisu nagu digidoc formaat ette näeb või ainult olemasolu nagu XML-DSIG ette näeb. Seega tunnistatakse allkirjad nüüd õigeks kui atribuudil on sisu aga valideerimisel nõutakse ka sisu vastavust formaadile.
- Käsureautiliidile lisati -ddoc-validate ja -cdoc-validate käsud.

Erinevused JDigiDoc versioonide 2.1.6 ja 2.1.1 vahel

- Parandati digidoc parserit UTF-8 formaadis failinime kuvamisel. Viga esines vaid win32 arvutites.
- Parandati BouncyCastleNotaryFactory.getNotaryCert() OCSP responderi

sertifikaadi valikut. Eelmine versioon ei kontrollinud mitme responderi sertifikaadi puhul korrektselt OCSP vastuse allkirja kehtivust iga sertifikaadiga. Nüüd kontrollitakse mitme responderi sertifikaadi puhul iga registreeritud antud responderi sertifikaadiga kuni ühega neist OCSP vastuse allkiri verifitseerub. Mitme responderi sertifikaadi registreerimine käib vastavalt järgmisele näitele:

```
DIGIDOC_OCSP2_CN=KLASS3-SK OCSP RESPONDER
DIGIDOC_OCSP2_CERT=jar://certs/sk-klass3-ocsp-responder-2006.cer
DIGIDOC_OCSP2_CERT_1=jar://certs/sk-klass3-ocsp-responder.pem
DIGIDOC_OCSP2_CA_CERT=jar://certs/sk-klass3.pem
DIGIDOC_OCSP2_CA_CN=TEST-SK
```

- Lisati CA sertifikaatide loetelusse: SK-EID sertifikaat ja OCSP responderi sertifikaatidesse uued "KLASS3-SK OCSP RESPONDER" ja "EID-SK OCSP RESPONDER" sertifikaadid. Vanad sertifikaadid on endiselt vajalikud varem allkirjastatud dokumentide allkirjade kontrollimisel. Teek valib automaatselt õige sertifikaadi vastavalt OCSP vastuse allkirjale.

JDigiDoc käsureautiliit

Teegi testimiseks on loodud ka eraldi käsureautiliit mille abil on võimalik kasutada enamust eetgi poolt pakutavatest võimalustest.

Üldised käsud

- **-? või -help** – kuvab abiinfot käskluste süntaksi kohta
- **-config <configuration-file>** - võimaldab edastada JDigiDoc konfiguratsioonifaili nime. Vaimikisi jar://jdigidoc.cfg.
- **-check-cert <certficate-file-in-pem-format>** - Kontrollib soovitud sertifikaadi staatust OCSP päringu abil.

Digiallkirjastamise käsud

- **-ddoc-in <input-digidoc-file>** - võimaldab edastada sisend digidocfaili nime.
- **-ddoc-new [format] [version]** – loob uue digidoc dokumendi objekti soovitud formaadi ja versiooniga. Vaikimisi formaadi versiooniks on 1.3 (uusim).
- **-ddoc-add <input-file> <mime-type> [content-type]** – lisab andmefaili uuele digidoc dokumendile
- **-ddoc-sign <pin-code> [manifest] [country] [state] [city] [zip]** – allkirjastab digidoc dokumendi.
- **-ddoc-out <ouput-file>** - kirjutab loodud või muudetud digidoc dokumendi soovitud faili.
- **-ddoc-list** – kuvab sisseloetud või loodud digidoc dokumendi andmefailide ja allkirjade infot. Kontrollib allkirju.
- **-ddoc-validate** – valideerib dokumendi ja kuvab võimalikud formaadi vead.

Krüpteerimise käsud

- **-cdoc-in <input-encrypted-file>** - võimaldab edastada sisendfaili nime
- **-cdoc-list** – kuvab krüpteeritud dokumendi andmeid (vastuvõtjad ja krüpteeritud andmeobjekt)
- **-cdoc-validate** – valideerib krüpteeritud dokumendi ja kuvab võimalikud formaadi vead.
- **-cdoc-recipient <certificate-file> [recipient] [KeyName] [CarriedKeyName]** – lisab

ühe vastuvõtja andmed loodavale krüpteeritud dokumendile.

- **-cdoc-encrypt <input-file> <output-file>** - krüpteerib andmed ja kirjutab loodud krüpteeritud dokumendi väljundfaili.
- **-cdoc-encrypt-stream <input-file> <output-file>** - krüpteerib sisendfaili ja kirjutab väljundfaili.
- **-cdoc-decrypt <pin> <output-file>** - dekrüpteerib sisseloetud krüpteeritud dokumendi.
- **-cdoc-decrypt-stream <input-file> <recipient> <pin> <output-file>** - dekrüpteerib sisendfaili ja kirjutab andmed väljundfaili.