TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut
Võrgutarkvara õppetool

Digitaalallkirjastamine võrgurakendustes Eesti ID-kaardi näitel

Bakalaureusetöö

Üliõpilane: Kristen Gilden

Üliõpilaskood: 103723IASB

Juhendaja: Tanel Tammet

Tallinn
2014

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem kellegi teise poolt kaitsmisele esitatud.

....

(kuupäev) (lõputöö kaitsja allkiri)

Annotatsioon

Käesoleva bakalaureusetöö „Digitaalallkirjastamine võrgurakendustes Eesti ID-kaardi näitel“ eesmärgiks on tutvustada Eesti ID-kaardiga digitaalallkirjastamise võimalusi võrgurakendustes. Teiseks eesmärgiks on lihtsustada allkirjastamise protsessi võrgurakendustes.

Töö tulemusena valmis avatud lähtekoodiga allkirjastamise liidese esimene arendusversioon *PHP* skriptikeeles. Lisaks sellele arendati liidese kasutamist illustreeriv võrgurakendus *Silex* mikrokarkassil.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 5 peatükki, 7 joonist ja 1 tabelit.

Abstract

Digital Signing in Web Applications Using Estonian ID-card

The thesis „Digital signing in web applications using Estonian ID-card“ is an attempt to introduce the numerous possibilities of digital document signing using Estonian ID-card in the context of web applications. The secondary aim of this work is to further simplify the use of ID-card signing in web applications.

As a result of this thesis an open source application programming interface for digital signing was released in the *PHP* scripting language. Additionally an example Silex web application was developed to illustrate the usage of said *API*.

The thesis is written in Estonian and contains 42 pages of text, 5 chapters, 7 figures and 1 table.

Sisukord

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides olemasoleva programmiga suhtlemiseks.
BDOC	Eestis kasutusele võetav digitaalallkirjastamise failivorming, *DDOC* edasiarendus.
CRL	*Certificate revocation list*, tühistatud avaliku võtme sertifikaatide nimekiri.
DDOC	Eestis kasutusel olev digitaalallkirjastamise failivorming.
HTTP	*Hypertext Transfer Protocol* ehk hüperteksti edastusprotokoll teabe edastamiseks arvutivõrkudes.
ID-kaart	Eesti Vabariigis kodanikele välja antav kiipkaart elektrooniline isikut tõendav dokument.
MVC	*Model View Controller*, enamlevinud võrgurakenduste arhitektuur.
OCSP	*Online Certificate Status Protocol*, protokoll sertifikaatide kehtivuse kontrollimiseks üle võrgu.
PHP	*PHP: Hypertext Preprocessor*, enamlevinud skriptikeel dünaamiliste võrgurakenduste arendamiseks.
RSA	Üks esimesi praktilisi avaliku võtme krüptosüsteem andmete turbeks.
Silex	*PHP* mikrokarkass, mille abil on võimalik süsteemide arenduskiirust tõsta.
SOAP	*Simple Object Access Protocol*, protokoll struktureeritud andmete edastamiseks võrguteenustes.
SSL	*Secure Socket Layer*, krüptograafiline protokoll andmete krüpteeritud edastamiseks arvutivõrkudes, *SSL* eelkäija.
TLS	*Transport Layer Security*, krüptograafiline protokoll andmete krüpteeritud edastamiseks arvutivõrkudes.
WSDL	*Web Service Description Language*, *XML*-markeeringus formaat võrguteenuste automaatseks tarbimiseks ja kasutamiseks.
XML	*Extensible Markup Language*, kirjelduskeel, mis on loetav arvutitele ja inimestele.

Sissejuhatus

Dokumentide digitaalne allkirjastamine on Eestis üha enam eelistatud tehingute tegemise viis. See on mugav, kiire ning turvaline. Nii allkirjastamine kui ka mitmed teistsugused e-teenused on võimalikud tänu ID-kaardile ja selle põhjal ehitatud taristule, mis nüüdseks on kasutusel juba üle kümne aasta.

ID-kaardi üsna pikka kasutusaega arvestades võiks eeldada, et oluliselt suurem hulk tooteid ja teenuseid on kiipkaardi võimalusi ära kasutanud. Ometi on kaart kasutusel pigem riigi ja suuremate ettevõtete võrgurakendustes. Üheks põhjuseks, miks ID-kaardi levik ei ole väga lai, võib olla vajalike tööriistade, teekide ja dokumentatsiooni vähene kättesaadavus ja madal kvaliteet. Tehnoloogia kiire leviku eelduseks on selle juurutamise lihtsus.

Antud bakalaureusetöö eesmärgiks on lihtsustada ID-kaardiga digitaalallkirjastamise protsessi *PHP* skriptikeeles, milles suur osa võrgurakendusi kirjutatud on. Selle eesmärgi saavutamiseks pannakse alus avatud lähtekoodiga digiallkirjastamise programmeerimisliidese projektile. Liidese kasutamise selgitamiseks arendatakse selle baasil lihtne allkirjastamise näiterakendus.

Töö on jaotatud viieks peatükiks, millest esimeses käsitletakse lähemalt olemasolevaid lahendusi: nii teeke kui ka võrguteenuseid ja valmislahendusi. Teises peatükis selgitatakse lähemalt digiallkirjastamise põhimõtteid abstraktsel tasandil. Kolmandas peatükis kirjeldatakse, kuidas täpsemalt töötab ID-kaardiga allkirjastamine. Neljas ja viies peatükk kirjeldavad vastavalt loodud näiterakendust ja selle jaoks arendatud programmeerimisliidest.

1. Olemasolevad lahendused

Eesti ID-kaardiga dokumentide allkirjastamiseks võrgurakendustes on juba olemas üsna lai valik näidiseid ja lahendusi. Käesolev peatükk annab ülevaate enamkasutatavatest lahendustest.

1.1. Sertifitseerimiskeskuse näidisrakendus

Sertifitseerimiskeskuse näidisrakendus asub aadressil https://www.openxades.org/ddservice/. Rakendus võimaldab üle võrgu *DigiDoc* konteinerite loomist, nende lugemist ja muutmist. Allkirjade lugemiseks kasutatakse lehitsejasse paigutatud pluginat. Rakendus on kirjutatud *PHP* skriptikeeles.

Näidisrakendus on ehitatud monoliitsena ja seetõttu pole uute süsteemide arendamisel sellest erilist kasu peale üldiste põhimõtete järgimise. Kuigi näidisrakenduses on realiseeritud *DigiDoc* klass, mida võiks ka teistes süsteemides kasutada, siis paraku on selle sõltuvused iganenud ega ühildu uuemate *PHP* versioonidega.

1.2. Eesti ID-kaardi rakendusjuhendi näiterakendus

Riigi Infosüsteemide Ameti projekti raames koostatud rakendusjuhendis leidub näiterakendus avalduste loomiseks [1]. Kujuteldava asutuse kliendid esitavad üle võrgu avaldusi, mis on digitaalselt allkirjastatud. Selleks pöörduvad kasutajad lehitseja abil võrguserveri poole, esitavad oma avalduse ning allkirjastavad selle. Seejärel saadab server allkirjastatud avalduse teisele teenusele, mis kontrollib allkirja ning allkirjastab avalduse omakorda asutuse digitempliga. Viimaks krüpteerib teenus dokumendi nii, et ainult antud kasutaja saab seda avada, ja tagastab selle. Lisaks on arendatud ka kasutaja arvutisse paigaldatav programm, mis otse allkirjastava teenusega suhtleb.

Kasutajaga otse suhtlev rakendus kasutab sama *DigiDoc* klassi, mida Sertifitseerimiskeskuse näidisrakendus, ent avalduste töötlemise teenuse jaoks on arendatud *PHP* mähis *DigiDoc* C teegile. Mähis võimaldab kasutada *DigiDoc C* teegi funktsionaalsust *PHP* andmetüüpide ja süntaksiga. Paraku on mähis mõeldud ainult näidises kasutamiseks ega võimalda kogu *C* teegi funktsionaalsust *PHP*-s kasutada. Kuna mähis ei kasuta *PHP* sisemisi andmestruktuure, ei ole automaatsel mäluhalduril aimu eraldatud mälust ning ebakorrektse kasutamise puhul võib rakendus mälu lekkida.

1.3. Teegid

Üks allkirja andmise võimalikke viise klient-server arhitektuuriga süsteemides on kasutada AS Sertifitseerimiskeskuse arendatud teeke. Käesoleval hetkel on teegid saadaval programmeerimiskeeltes *Java, C, C++, .NET* ja *Visual Basic*. Kahtlemata saab neid suuremal või

vähemal määral võrgurakenduses kasutada. Käsitleme lähemalt *C* versiooni allkirjastamise teegist, kuna enamik üldkasutatavatest keeltest toetab *C* teekide sidumist.

C teek võimaldab kasutajal koostada digitaalselt allkirjastatud dokumente, nende õiguspärasust kontrollida ning ka dokumente krüptida [2 lk 5]. Enne igasuguse toimingu alustamist tuleb teek algseadistada. Seejärel ehitatakse mälus vastavalt vajadusele uue või olemasoleva allkirjastatud dokumentide konteineri struktuur. Viimaks genereeritakse koostatud struktuuri järgi reaalne *DigiDoc* fail ja kirjutatakse see failisüsteemi. Tuleb mainida, et käesoleval hetkel on toetatud ainult *DDOC* failivormingu versioon 1.3 ja seega *BDOC* vormingus konteinereid luua pole võimalik. Tühja konteineri näidis on toodud lisas (Lisa 4 – DigiDoc C teegi kasutamise näidiskood).

Võttes arvesse, et võrgurakendused on oma olemuselt sunnitud suhtlema välismaailmaga, ei ole kindlasti mõistlik *C* teeki rakenduses otse kasutada. Teegi kasutamine on keeruline, kuna pakutav dokumentatsioon jätab sageli soovida. Puudub selge ülevaade funktsioonidest, mis on mõeldud väliseks kasutamiseks ning mis sisemiseks. See võib põhjustada hiljem probleeme teegi uuendamisel, kuna kasutatud funktsionaalsust ei pruugi enam eksisteeridagi.

Siinkohal tuleb märkida, et uute süsteemide arendamisel soovitab AS Sertifitseerimiskeskus eelistada *Java* teeki *JDigiDoc*. Lisaks teegile pakutakse ka käsurea utiliiti, mille kaudu peaks teoorias olema võimalik mistahes keskkonnast *Java* teegiga suhelda. Kahjuks on utiliidi kaudu võimalik dokumente allkirjastada ainult nii, et ID-kaart asub füüsiliselt samas arvutis. Seetõttu pole utiliiti võimalik võrgurakendustes kasutada. *Java* teeki ennast saab kasutada vaid *Java* programmeerimiskeeles kirjutatud rakendustes.

1.4. SOAP võrguteenus

AS Sertifitseerimiskeskus pakub *SOAP* protokollil põhinevat võrguteenust *DigiDocService*, mille abil on võimalik läbi viia isikutuvastamine, digiallkirjastamist ja allkirjade tuvastamist nii ID-kaardi kui ka Mobiil-ID vahendusel [3]. Sisuliselt tähendab see seda, et allkirjastamisega seotud toimingud tehakse üle võrgu eelpool mainitud teekidega. Seega pole teeke serverisse otseselt paigaldada tarvis.

1.4.1. SOAP

SOAP ehk *Simple Object Access Protocol* on mõeldud struktureeritud informatsiooni vahetamiseks jaotatud, hajutatud süsteemides [4]. Protokolli formaadiks on *XML* ja seega on see kasutatav üle mitmesuguste erinevate alusprotokollide, peamiselt aga üle *HTTP*. Tüüpiline *SOAP*-päring näeb välja järgnev.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2014-06-18T14:00:00-03:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:teadaanne xmlns:m="http://example.org/alert">
   <m:msg>See on näidis.</m:msg>
  </m:teadaanne>
 </env:Body>
</env:Envelope>
```

1.4.2. Kasutamine võrgurakendustes

Teenuse kasutamiseks võrgurakendustes on tarvis teenusega suhtlevat *SOAP* klienti. Erinevate keskkondade jaoks leidub hulgaliselt mitmesuguseid vabavaralisi *SOAP* teeke. Kõige mõistlikum on *PHP* puhul kasutada *SOAP* laiendit, mis uuematel versioonidel on juba *PHP* paketti integreeritud.

Iga uue dokumendi või nende kogu allkirjastamiseks või juba olemasoleva *DigiDoc* konteineri avamiseks tuleb teenusega avada uus sessioon. Seejärel saadetakse võrguteenusele ükshaaval allkirjastatavad dokumendid ja allkirjastajate sertifikaadid. Sertifikaatide ja dokumentide alusel koostatud räsi tuleb saata lehitsejale, et seal asuva plugina kaudu saaks ID-kaart signatuuri arvutada. Seejärel saab allkirjad kinnitada. Viimaks on serveril võimalik võrguteenuses asuv konteiner alla laadida ning seda oma äranägemise järgi edasi töödelda.

DigiDoc teenuse kasutamine võrgurakendustes on küllaltki tülikas. Lisaks *SOAP* protokolli tundmisele tuleb peensusteni kursis olla teenuse kasutamise eripäradega. Näiteks tuleb hoolitseda ise selle eest, et dokumente ei lisataks peale seda, kui konteineri küljes on juba vähemalt üks allkiri. Võttes arvesse, et teenuse kasutamine on tasuline, tasub kindlasti kaaluda alternatiive.

2. Digiallkirjastamise põhimõtted

Võrreldes paberkandjal antava allkirjaga peab digitaalne allkiri täitma mitmeid täiendavaid kriteeriume. Kuna digitaalsete andmete kopeerimine ei nõua sisuliselt mingit vaeva, peab digitaalne allkiri olema nõnda sõnumi ja allkirjastajaga seotud, et allkirjastatud sõnumi muutmisel allkiri enam ei kehtiks.

Digitaalse allkirjastamise süsteemid koosnevad kolmest põhilisest komponentidest [5 lk 283]:

- Algoritm võtmete genereerimiseks;
- Allkirjastamise algoritm, mille abil saab sõnumi allkirjastada;
- Allkirja kontrollimise algoritm, millega on võimalik sõnumi allkirja verifitseerida;

Käesolev peatükk käsitleb lähemalt allkirjastamise põhimõtteid kui asümmeetrilise krüptograafia pöördtehet.

2.1. Asümmeetriline krüptograafia

Asümmeetriliseks krüptograafiaks nimetatakse sellist liiki krüptograafilisi algoritme, milles andmete krüptimiseks ja dekrüptimiseks kasutatakse vastavalt avalikku ja salajast võtit ehk võtmepaari. Sõnum, mida soovitakse edastada, krüptitakse vastuvõtja avaliku võtmega šiffertekstiks. Selle dekrüptimine esialgseks sõnumiks on võimalik vaid avaliku võtmega seotud salajase võtmega. Erinevus sümmeetrilisest krüptograafiast seisnebki mõlema tehte jaoks erineva võtme kasutamises.

2.1.1. Diffie-Hellman’i võtmevahetus

Üheks esimestest asümmeetrilistest krüptograafia süsteemidest võib pidada Diffie-Hellman’i võtmevahetust. Võtmete tuletamiseks kasutatakse diskreetse logaritmi probleemi omadusi. Osapooled valivad ühised aluse α ning algarvu q. Iga osapool genereerib juhusliku arvu X_i, mille jätab avalikustamata. Seejärel arvutab igaüks avaliku saladuse vastavalt valemile

$$Y_i = \alpha^{X_i} \bmod q$$

Kui osapooled i ja j soovivad omavahel salajaselt suhelda, arvutavad nad ühise saladuse

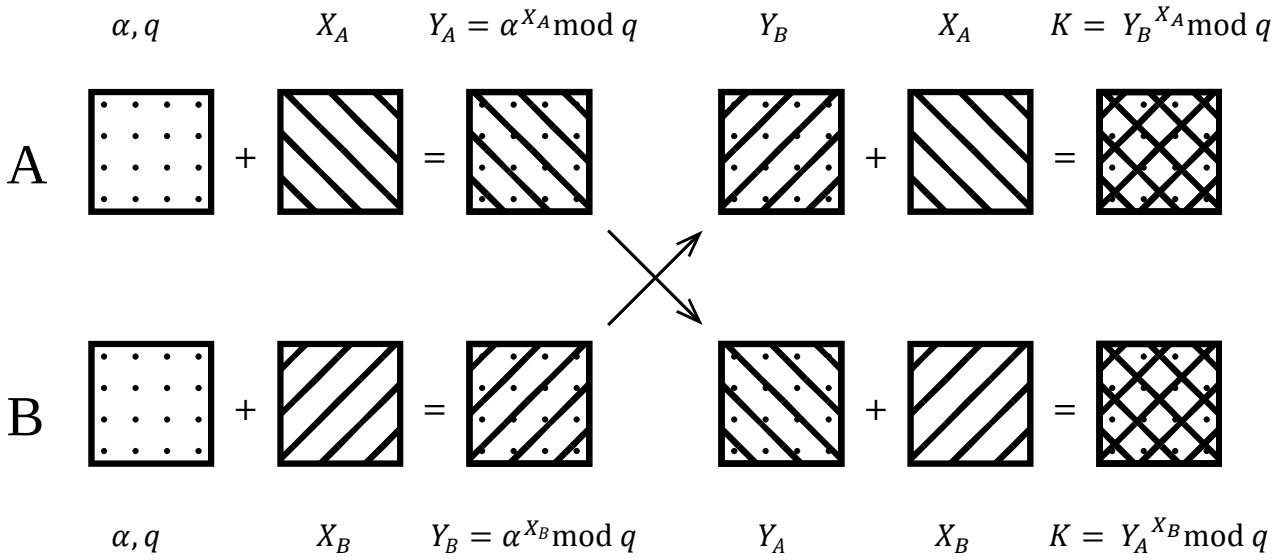
$$K_{ij} = \alpha^{X_i X_i} \bmod q$$

Osapool i jõuab antud tulemuseni kasutades osapoole j avalikku võtit Y_j, saades

$$K_{ij} = {Y_j}^{X_i} \bmod q = (\alpha^{X_j})^{X_i} \bmod q = \alpha^{X_j X_i} \bmod q$$

Analoogselt saab osapool j sama tulemuse, arvutades $K_{ji} = {Y_i}^{X_j} \bmod q = \alpha^{X_i X_j} \bmod q$. Kolmas osapool peaks ühise saladuse arvutamiseks tegema tehte $K_{ij} = {Y_i}^{(\log_\alpha Y_j)} \bmod q$, mis arvutuslikult on aga eksponentsiaalselt keerukas. [6 lk 649]

Eelnevat saab illustreerida järgneva skeemiga (Joonis 1). Eeldatud on, et mustrite üksteisest eraldamine on mittetriviaalne. Ühisele mustrile liidavad osapooled oma salajase mustri, saadud tulemus vahetatakse ning seejärel lisatakse vahetatud mustrile taaskord salajane muster, jõudes identse tulemuseni.



Joonis 1. Diffie-Helman'i võtmevahetus mustrite liitmise näitel.

2.2. RSA krüptosüsteem

1977. aastal esitlesid Ron Rivest, Adi Shamir ja Lenoard Adleman tol ajal uudset asümmeetrilist krüptosüsteemi, mis pälvis hiljem nime RSA. Selle algoritmi abil on võimalik genereerida avalik ja salajane võtmepaar nõnda, et avaliku võtmega krüptitud sõnum on võimalik dekrüptida salajase võtmega. Erinevus Diffie-Hellmani võtmevahetusest seisneb selles, et RSA võimaldab täielikult asümmeetrilist krüptimist, Diffie-Hellmani võtmevahetus aga jagatud saladuse vahetamist, mida saab hiljem kasutada näiteks võtmena sümmeetrilises krüptosüsteemis.

Võtmete genereerimine põhineb suurte algarvude teguriteks jaotamise keerukusel. Esmalt arvutatakse arv n korrutisena kahest suurest juhuslikust algarvust p ja q. Seejärel valitakse suur juhuslik täisarv d salajaseks võtmeks nii, et arvust n väiksemate ühise tegurita arvude summa (Euleri funktsioon) ja arvu d suurim ühistegur oleks 1, see tähendab

$$\text{SÜT}\big(d, \varphi(n)\big) = \text{SÜT}\big(d, (p-1)(q-1)\big) = 1$$

Viimaks arvutatakse avalik võti e pöördarvuna salajasest võtmest d moodul n nii, et

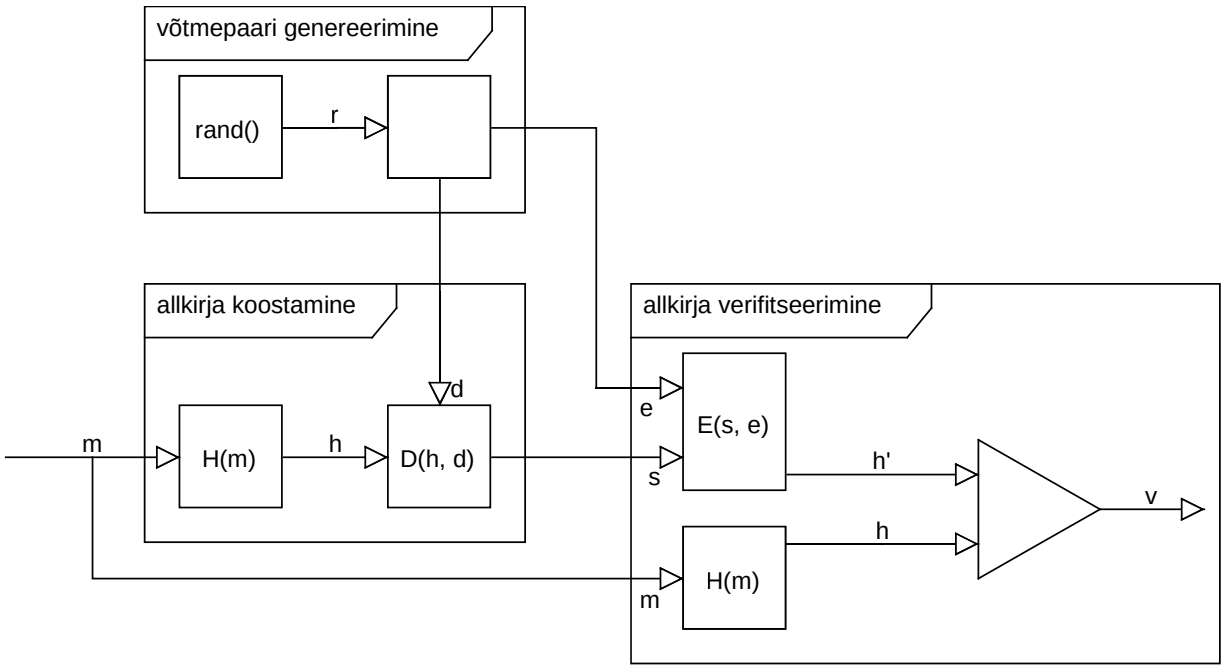
$$e \cdot d \equiv 1 \left(\mathrm{mod}\ (p-1)(q-1)\right)$$

Olgu krüptitav sõnum m, siis krüptimis- ja dekrüptimisfunktsioon on vastavalt $E(m,e) = m^e \bmod n$ ja $D(m,d) = m^d \bmod n$. [7 lk 122-123]

2.2.1. Sõnumi allkirjastamine

Osutub, et allkirjastamine on teatud mõttes krüptimise pöördtehe. Olgu allkirjastajal genereeritud võtmepaar e (avalik) ja d (salajane). Allkirjastaja arvutab allkirja s sõnumile m oma salajase võtmega $s = D(m,d)$, dekrüptides sisuliselt avatekst suvaliseks baidijadaks. Teine osapool saab allkirja õigsust kontrollida, krüptides avaliku võtmega baidijada tagasi avatekstiks ehk m= $E(s,e) = E(D(m,d),e)$ [7]. Kui edastav sõnum ja allkirjast arvutatud tulemused on omavahel võrdsed, saabki kontrollija olla veendunud allkirja õigsuses.

Reaalsetes süsteemides arvutatakse allkiri sõnumi asemel krüptograafilise räsifunktsiooni H pealt. Seega saab sõnumi allkirja õigsuses veenduda, leides et $H(m) = E(D(H(m),d),e)$. Eeliseks on allkirja fikseeritud pikkus ja nii ei tule allkirja tõttu saata esialgsest sõnumist kaks korda rohkem andmeid. Räsi pealt arvutatava allkirjastamise viisi kasutavad näiteks OpenPGP [8 lk 6] ja DSA [9 lk 15]. Alljärgnev diagramm illustreerib tervet protsessi – nii allkirjastamist kui ka selle kontrollimist (Joonis 2).



Joonis 2. Digitaalne allkirjastamine. Genereeritakse võtmepaar, salajase võtmega d arvutatakse sõnumi m räsist h allkiri s. Allkirja verifitseerimiseks kasutatakse krüptimisfunktsiooni avaliku võtmega ja leitakse h'. Kui h' on võrdne sõnumi m räsiga h, on allkiri järelikult antud salajase võtmega d.

2.3. Allkirja kontrollimine

Kuidas veenduda, et digitaalse allkirja on andnud just see isik, kes ta väidab end olevat? Kõige vahetum viis selle tuvastamiseks oleks füüsiliselt kohtuda ja veenduda, et isiku avalik võti kuulub tõesti temale. Ilmselgelt muudaks selline nõue digitaalse allkirjastamise kui sellise tarbetuks. Aeg-ajalt tekib ka vajadus võtmepaare välja vahetada. Seega on tarvis kuidagi ka allkirjade andmisega siduda võtmete kehtivus.

2.3.1. Avaliku võtme sertifikaadid

Enamlevinud viisiks siduda mõni isik kindla võtmega on kasutada avaliku võtme sertifikaate. Sertifikaadil on kirjas isiku avalik võti ja mõne teise isiku digitaalne allkiri, mis tõendab et antud avalik võti kuulub mainitud isikule. Võimalik on kasutada nii keskendatud kui hajutatud süsteemi.

Hajutatud süsteemi puhul võib sertifikaadi allkirjastajaks olla mistahes isik eeldusel, et ta on usaldusväärne allkirja kontrollija silmis ning kontrollijal on eelnevalt saadud tema avalik võti. Sellist skeemi kasutavad näiteks krüptoprogrammid *PGP, OpenPGP* ja teised. Hajutatud lahenduse puhul on igal kasutajal suurem kontroll selle üle, keda usaldada ja keda mitte. Samas on avalike võtmete vahetamine tülikas, sest eeldab füüsilist kontakti.

Keskendatud sertifikaatide kontrollimise süsteemi puhul on valitud väike hulk organisatsioone, sertifikaadi autoriteete, kelle väljaantud sertifikaate usaldatakse. Nende asutuste ise allkirjastatud sertifikaadid ehk juursertifikaadid jõuavad lõppkasutaja arvutisse tavaliselt erinevate programmide paigaldamisega. Mõne sertifikaadi kontrollimiseks peab kasutaja sertifikaatide ahelat pidi üles liikuma, kuni leiab lõpuks sertifikaadi, mis tema arvutisse juba paigaldatud on. Selline meetod on enamlevinud ning ühtlasi kasutusel nii *TLS* protokollis võrguressursside tuvastamiseks kui ka Eesti ID-kaardi süsteemis. Viimase puhul tagab iga kodaniku seotuse kindla võtmepaariga AS Sertifitseerimiskeskuse juursertifikaat.

2.3.2. Kehtivuskinnitus

Lisaks isiku kokkuviimisele võtmepaariga peab allkirja kontrollimiseks olema võimalik veenduda, et antud võtmed olid allkirjastamise hetkel kasutusel. See tähendab, et võtmeid peab olema võimalik kuidagi tühistada. See vajadus võib tekkida näiteks olukorras, kus kasutaja salajane võti saab kompromiteeritud või osutub, et kasutuselolevad krüptograafilised algoritmid ei ole enam turvalised.

Allkirja andmisel kasutaja sertifikaadi kehtivuse kontrollimiseks on mitu võimalust. Tüüpiliselt kasutatakse keskendatud sertifikaatide kontrollimise skeemi puhul praktikat, kus sertifikaadi autoriteet koostab nimekirja tühistatud sertifikaatidest ja avalikustab selle [10 lk 11]. Tühistatud

sertifikaatide nimekiri ehk *Certificate Revocation List* tuleb igal teenuse kasutajal seega perioodiliselt oma nimekiri sertifikaadi autoriteediga sünkroniseerida.

Kuna digitaalse allkirjastamise puhul on oluline, et kasutatav sertifikaat oleks allkirja andmise hetkel kehtiv, siis *CRL* süsteemist ei piisa. Kehtivuse kontrollimiseks saab tühistatud sertifikaatide nimekirja asemel kasutada *Online Certificate Status Protocol* ehk lühidalt *OCSP* päringut. See protokoll võimaldab teha teenusepakkujale mõne kindla sertifikaadi kohta päringu, mis tagastab, kas see on tühistatud või mitte [11 lk 1].

3. Eesti ID-kaardiga digiallkirjastamise reaalsed viisid

Eesti Vabariigis antakse kehtiva isikutunnistuse välja kiipkaarte, mida nimetatakse ID-kaartideks. Kiibis on nii kodaniku isikut tuvastavad andmed kui ka nende andmete õigsust kinnitavat sertifikaadid. Täpsemalt sisaldab ID-kaart kahte salajast võtit ja neile vastavat sertifikaati avalike võtmetega [12 lk 35], kusjuures kasutusel on eelnevalt kirjeldatud RSA krüptosüsteem. Esimest sertifikaati on võimalik kasutada autentimiseks, teine on aga mõeldud digiallkirjade andmiseks.

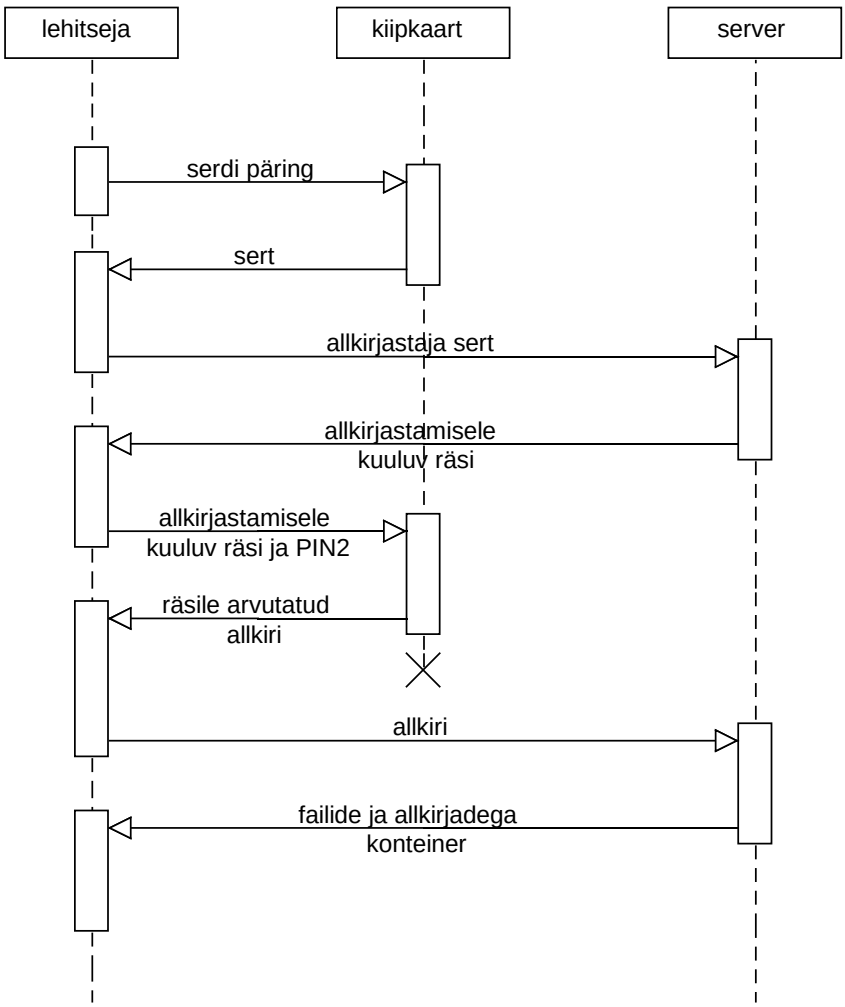
3.1. Allkirjastamine personaalarvutis

Kõige lihtsam viis dokumentide allkirjastamiseks on kogu protsess läbi viia kohalikus arvutis. Allkirjastamiseks on esmalt tarvis kiipkaardilugejat, seejärel tuleb paigaldada spetsiaalne ID-kaardi utiliit. Allkirjastamiseks märgitakse ära soovitud failid ning sisestatakse PIN-kood krüptimiskäsu edastamiseks kaardile. Utiliit saadab allkirjastatava räsi kaardilugeja vahendusel kiipkaardile, millest seejärel salajase võtmega allkiri arvutatakse. Saadud tulemus paigutatakse koos avaliku võtme sertifikaadi ja failidega ühte konteinerisse, mis seejärel kasutaja arvuti kõvakettale salvestatakse. Allkirjastatud dokument või nende kogu on võimalik huvitatud osalistele saata suvalise sidekanali kaudu.

3.2. Võrgurakendus kui allkirjastamise vahendaja

Keerukamate nõuete jaoks on mõistlikum kasutada alternatiivina keskendatud võrgulahendust. See võimaldab enamuse töövoost automatiseerida nii, et lõppkasutaja peab sekkuma vaid allkirja andmisel. Üheks võimalikuks kasutusvaldkonnaks on näiteks lepingu digitaalne sõlmimine, kus leping genereeritakse kliendile automaatselt.

Andmete allkirjastamine võrgurakenduse kaudu on tunduvalt keerulisem, nõudes mitmete komponentide ja tehnoloogiate koostööd. Allkirjastaja koos kiipkaardiga ja allkirjastatavad andmed asuvad erinevates arvutites. Lisaks sellele pole võimalik kasutaja võrgulehitsejast suhelda kiipkaardi lugeja kui riistvaralise seadmega turvapiirangute tõttu. Niisiis on kogu allkirjastamise töövoog veidi teistsugune. Võrgurakendus saadab kasutaja võrgulehitsejale allkirjastatavate dokumentide räsi. Lehitsejas käivitub skript, mis palub plugina abil räsi allkirjastada. Plugin avab modaalakna PIN-koodi sisestamiseks. Seejärel saadab plugin räsi kaardilugejale, mis kiipkaardi poolt viimaks allkirjastatakse. Saadud allkiri saadetakse serverile, mis seejärel ise allkirjastatud konteineri valmis ehitab (Joonis 3).



Joonis 3. Digiallkirjastamine võrgurakenduse kaudu. Järgnevusdiagrammilt on lihtsustamise nimel välja jäetud konteineri esialgne koostamine ning OCSP päringu tegemine.

Eelnevast tuleneb, et paraku on mõlemal juhul kasutajal siiski tarvis arvutisse paigaldada spetsiaalset tarkvara. Kohaliku allkirjastamise puhul tuleb installeerida ID-kaardi utiliit, võrgurakenduse puhul on tarvis kaardilugejaga suhtlevat võrgulehitseja pluginat.

3.3. DigiDoc konteineri failivormingud

Allkirjastatavate dokumentide ja nendele vastavate allkirjade sidumiseks ühtseks tervikuks kasutatakse spetsiaalselt failivormingut – konteinerit. Käesoleval hetkel on kasutusel kaks erinevat vormingut. Mõlemad vormingud põhinevad *XML Advanced Electronic Signatures (XAdES)* [13] standardil.

3.3.1. DDOC

DigiDoc ehk *DDOC* on *XML*-põhine failivorming digitaalsete allkirjade ja allkirjastatud dokumentide talletamiseks. *DDOC* vormingu omadusteks on allkirjade verifitseeritavus lisainfota, mitme dokumendi üheaegne allkirjastamine, formaadiründe vastane kaitse, võimalus hoida

algdokumente konteineri sees või sellest eraldi, mitme allkirja omistamise tugi ning iga allkirja kohta nõutav kehtivuskinnitus [14].

DDOC faili põhistruktuur on järgnev.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SignedDoc format="DIGIDOC-XML" version="1.3"
  xmlns="http://www.sk.ee/DigiDoc/v1.3.0#">
    <!-- Algfailid (1..n) -->
    <DataFile />
    <!-- Digiallkirjad (0..n) -->
    <Signature />
</SignedDoc>
```

Iga faili kohta on konteineris üks *DataFile* element, mis võib olla väärtustatud faili sisuga *base64* tekstkodeeringus. Atribuutide abil määratakse faili unikaalne järjenumber konteineris, sisutüüp, suurus ja salvestamise meetod. Neist viimase abil saab määrata, kas fail lisatakse kodeeritult konteinerisse või on selle asemel atribuutidena kaasa antud pelgalt faili räsi ja selle arvutamise algoritm.

Iga allkirja kohta on konteineris üks *Signature* element. See koosneb omakorda mitmesugustest allkirja defineerivatest elementidest. Neist olulisemad on signatuuri element *SignatureValue* ning signatuuri verifitseerimist võimaldav avalikku võtit sisaldav sertifikaat elemendina *KeyInfo*. Lisaks sellele sisaldab *SignatureValue* allkirjastaja sertifikaadi kehtivuskinnitusega seotud andmeid.

DDOC failivormingu näidise võib leida lisast (Lisa 2 – DDOC ja DDOC failivormingu näited).

3.3.2. BDOC

BDOC on uus digitaalallkirjastamise failivorming, mis peaks peagi asendama *DDOC* vormingu. See koosneb mitmest erinevast profiilist ja konteineri vormingu definitsioonist. Lisaks riikideülesele ühilduvusele seisneb uue vormingu eelis ka sissehitatud kokkupakitavuses. Et konteiner ei ole midagi muud, kui pelgalt *ZIP*-konteiner, peaks see vähendama ka probleeme allkirjastatud dokumentide saatmise e-maili teel.

Põhiprofiil on *XML*-struktuur, mis sisaldab üht krüptograafilist allkirja üle defineeritud andmekogumi [15]. See on aluseks järgnevatele profiilidele, millest esimesed kaks defineerivad allkirjastaja sertifikaadi kehtivuse sidumise allkirjaga.

Ajamärgendus ehk *time-marking* profiil nõuab vormingu rakendajatel võtta pärast allkirja loomist ühendust sertifikaadi väljaandja *OCSP* teenusega vastavat protokolli kasutades ja saadud vastus

XML-struktuuri juurde liita. Juhul kui *OCSP* teenus ei vasta spetsifikatsioonis esitatud nõuetele, tuleb lisada ajatempel ehk *timestamp*.

Säilitamaks allkirjade usaldusväärsuse pikemas perspektiivis, tuleb tagada nende vastupidavus arvutijõudluse kasvule ja krüptograafiliste algoritmide võimalikele turvalisuskadudele. Selleks on spetsifitseeritud kaks erinevat moodust. Sertifikaadi kehtivuskinnitusi väljastav teenusepakkuja võib pidada päringute logi, mille kirjed on avalikult kontrollitavad. Alternatiivselt võib võtta kasutusele arhivaalse ajatempli. Sel juhul tembeldatakse allkirjastatud dokumendid perioodiliselt üle. Mõistetavalt peaks selle toimimiseks olema kõik dokumendid keskendatud ühte süsteemi.

BDOC konteineri vorming põhineb *ZIP*-konteineril. Selles peab kindlasti leiduma sisutüübi fail nimega *mimetype* ja sisaldama *BDOC* sisutüüpi, milleks on *application/vnd.etsi.asic-e+zip*. Lisaks peab selles olema kõigi allkirjastavate failide nimekirja sisaldav manifesti fail nimega *manifest.xml*. Allkirjad asuvad kaustas *META-INF,* kusjuures iga põhiprofiilil baseeruv allkirja faili nimi peab sisaldama sõnet *signatures*. *BDOC* konteineri täpsem näidis on esitatud lisas (Lisa 2 – DDOC ja DDOC failivormingu näited).

4. Loodud näiterakendus arendajale

4.1. Arhitektuur

Näiterakendus järgib *Model-View-Controller* ehk lühidalt *MVC* mustrit, mida peetakse praktikas paindlikuks lahenduseks rakenduste ehitamisel [16 lk 118]. Seda põhjusel, et rakenduse mudel ehk äriloogika on eraldatud nii vaadetest kui ka kontrolleritest. Lihtsuse huvides on näiterakenduse äriloogika ühes failis (*/app.php*) ning selle alusel mallist genereeritav *HTML* teises (*/templates/index.php*). Taoline jaotus võimaldab koodi analüüsimisel paremini keskenduda allkirjastamise protsessi kui sellisesse, sest *HTML*-markeering on teises failis.

MVC realiseerimiseks on kasutatud *Silex*-i mikrokarkassi. Kõik päringud suunatakse läbi juurkaustas asuva /app.php faili. See peidetaks reaalses rakenduses näiteks *Apache* mooduliga *mod_rewrite*. Iga spetsiifiline tegevus on kapseldatud anonüümsesse funktsiooni, millega on vastavusse seatud kindel *URI* (universaalne ressursiidentifikaator). Alljärgnev näide tagastaks „Tere, maailm!“ aadressilt http://localhost/app.php/tere/maailm eeldusel, et see asub veebiserveri juurkaustas nime *app.php* all.

```
<?php
$app = new \Silex\Application();
$app->get('/tere/maailm', function () { return 'Tere, maailm!'; });
$app->run();
```

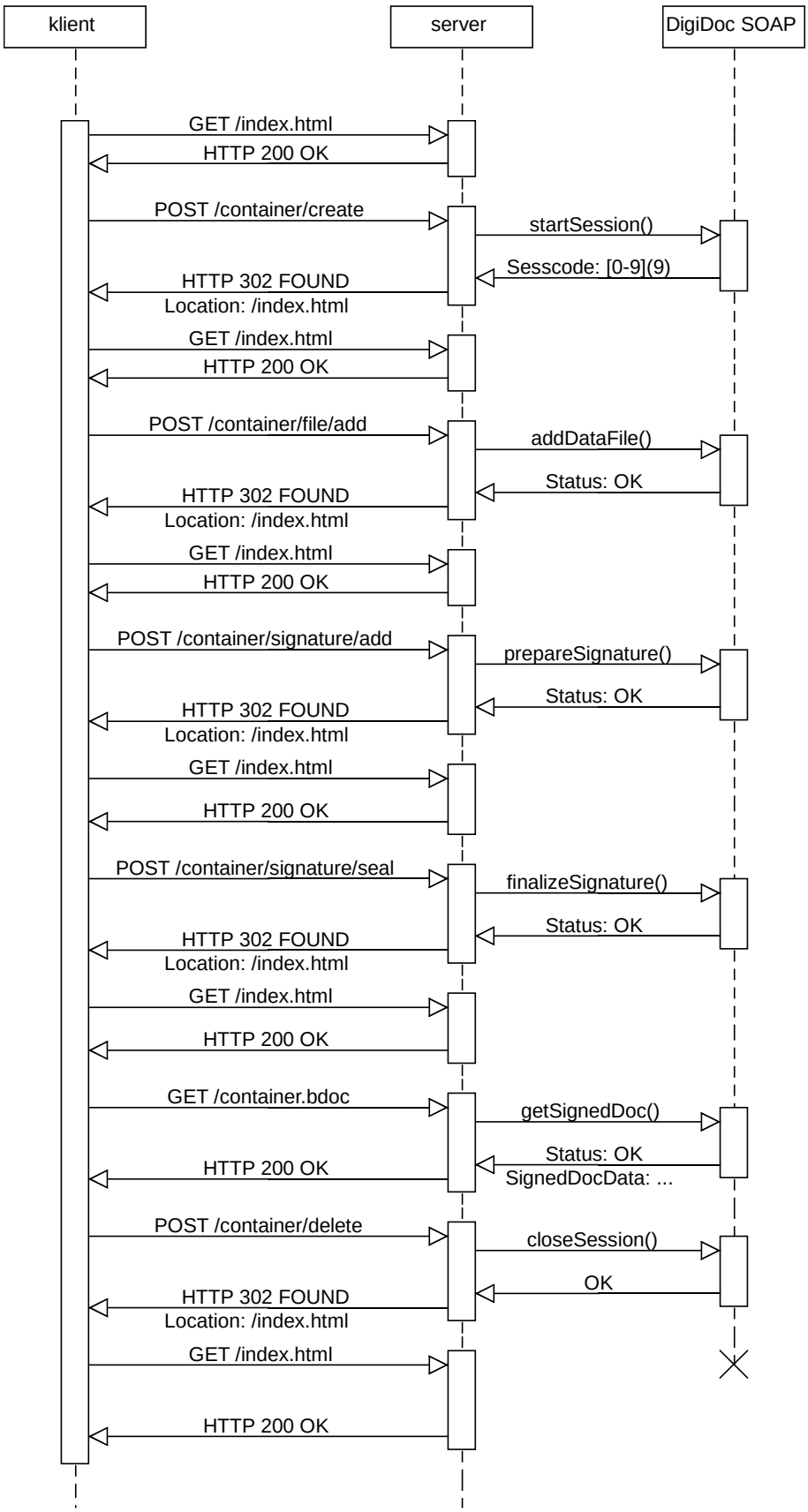
Näiterakenduses on mitu taolist *URI*-le vastavat anonüümset funktsiooni kasutusel (Tabel 1).

URI	Meetod	Parameetrid	Kirjeldus
/index.html	GET	–	Kuvab rakenduse oleku HTML formaadis. Selle aadressi kaudu on võimalik kõiki järgnevaid käske rakendada.
/container/create	POST	cert[id] cert[signature] file	Tekitab sessiooni uue konteineri ühe faili ja (kinnitamata) allkirjaga.
/container/delete	POST	–	Kustutab kasutaja sessioonist konteineri.
/container.bdoc	GET	–	Tagastab sessioonis asuva konteineri sisu sellisena, nagu see hetkel *DigiDoc* teenuses on.

/container/signature/add	POST	cert[id] cert[signature]	Lisab konteinerile ühe allkirja juurde.
/container/signature/seal	POST	signature[id] signature[solution]	Kinnitab allkirja.
/container/file/add	POST	file	Lisab konteinerile faili. Kui konteineril on vähemalt üks allkiri, siis antud meetod tagastab veateate.

Tabel 1. Näiterakenduse HTTP meetodid

DigiDoc konteineri loomine algab allkirjastatava faili valimisest ning kasutaja ID-kaardilt sertifikaadi seerianumbri ja signatuuri lugemisega. Viimane on võimalik brauseri *DigiDoc* pluginaga suheldes. AS Sertifitseerimiskeskus on tarninud lisaks pluginale JavaScripti teegi, mis peidab endasse erinevate lehitsejate iseärasused. Seda ka antud näiterakendus kasutab. Kasutaja sertifikaadi andmete kättesaamisel avab rakendus *DigiDoc* võrguteenusega sessiooni ning loob konteineri. Lisatud allkiri on hetkel veel kinnitamata – võrguteenuselt saadud konteineri räsi tuleb kuidagi edastada kasutaja ID-kaardile, et signatuur arvutada. Selleks tuleb taas läbi JavaScripti teegi brauseri pluginaga ühendust võtta, mis soovitud sõnumi kaardile allkirjastamiseks edastab ning saadud tulemuse tagastab. Viimase sammuna tuleks tulemus võrguteenusele edastada. Seejärel on võimalik võrguteenuselt allkirjastatud konteiner pärida. Kogu protsessi illustreerib järgnev joonis (Joonis 4).



Joonis 4. Allkirja andmine läbi veebirakenduse

4.2. Installimine ja seadistamine

Kõige lihtsam on näidisrakendust proovida mõnel GNU\Linux distributsioonil. Eelduseks on vähemalt programmide *openssl* ja *php* olemasolu. Kasutades *php* versiooni, mis on suurem kui 5.4.0, saab kasutada käsurealt käivitatavat sisseehitatud *HTTP* serverit – sel juhul piisab programmi *stunnel* installeerimisest ning täiendavat võrguserveri tarkvara tarvis ei ole. Järgnev juhis kasutabki *php* sisseehitatud serverit lühiduse huvides, ent loomulikult sobib ka mõni professionaalne lahendus nagu näiteks *Apache* või *nginx*.

Kui seda veel tehtud pole, tuleb esmalt genereerida salajane võti ning ise allkirjastatud sertifikaat. Alljärgnevas näite tulemusel koostatakse 365 päevaks 2048-bitise *RSA* võtmega sertifikaat, kusjuures salajane võti pannakse koos sertifikaadiga ühte faili *stunnel*-ile sobivas formaadis.

```
$ openssl –req –days 365 –nodes –newkey rsa:2048 –keyout stunnel.pem –out
stunnel.pem
```

Seejärel soovib *OpenSSL* sertifikaadi kontaktandmeid, mis antud rakenduse kontekstis ei oma tähtsust, seega need väärtused võivad olla suvalised. Järgnevalt tuleks lokaalselt installida näiterakenduse tööks vajalikud sõltuvused. Selleks tuleb kasutada *PHP* paketihaldurit *Composer*.

```
$ curl –sS https://getcomposer.org/installer | php
$ php composer.phar install
```

Järgmise sammuna tuleks alustada PHP server.

```
$ php –S localhost:8080
```

Viimaks tuleks käivitada *stunnel*, mis käitub proksina ja tagab *SSL/TLS* võimekuse. Ühendus peab olema turvatud. Vastasel korral lehitseja plugin keeldub töötamast.

```
$ stunnel –d 4443 –f –r 8080 –p stunnel.pem –P /tmp/stunnel.pid
```

Seejärel peaks lehitsejaga aadressile https://localhost:4443/app.php/index.html navigeerides avanema näiterakendus. Kuvatav *SSL/TLS* hoiatus on tavapärane, sest kasutatakse ise allkirjastatud sertifikaati.

5. Näiterakenduse jaoks loodud uus API

5.1. Kasutamine

Enne konkreetsete ülesannete lahendamist tuleb tekitada uus *Api* objekt, mis hakkab *DigiDoc* võrguteenusega suhtlema.

```
<?php
// võrguserveris

use KG\DigiDoc\Api;
use KG\DigiDoc\Soap\Client;
$api = new Api(new Client());
```

Vaikimisi kasutatakse *DigiDoc* võrguteenuse testversiooni. Reaalsetes rakendustest tuleks suunata *SOAP* klient õige teenuse aadressi poole.

```
<?php
// võrguserveris

$api = new Api(new Client(array(), 'https://digidocservice.sk.ee'));
```

5.1.1. Uue allkirjastatud konteineri loomine

Uue allkirjastatud konteineri loomiseks tuleb läbi käia teha üsna mitu kohustuslikku etappi. Esmalt tuleks luua uus tühi konteiner. Seejärel tuleb lisada soovitud allkirjastatavad failid ning siis uus allkirja objekt. Pärast *DigiDoc* teenuse uuendamist on allkirja objektil konteineri räsi, mis tuleb allkirjastamiseks saata kliendi lehitsejasse. Saanud tagasi allkirjastatud tulemuse, tuleb see omistada allkirja objektile. Järgmiseks tuleb kogu konteinerit taas uuendada. Seejärel saab allkirjastatud dokumendi salvestada failisüsteemile. Järgnev näide illustreerib eelnevat.

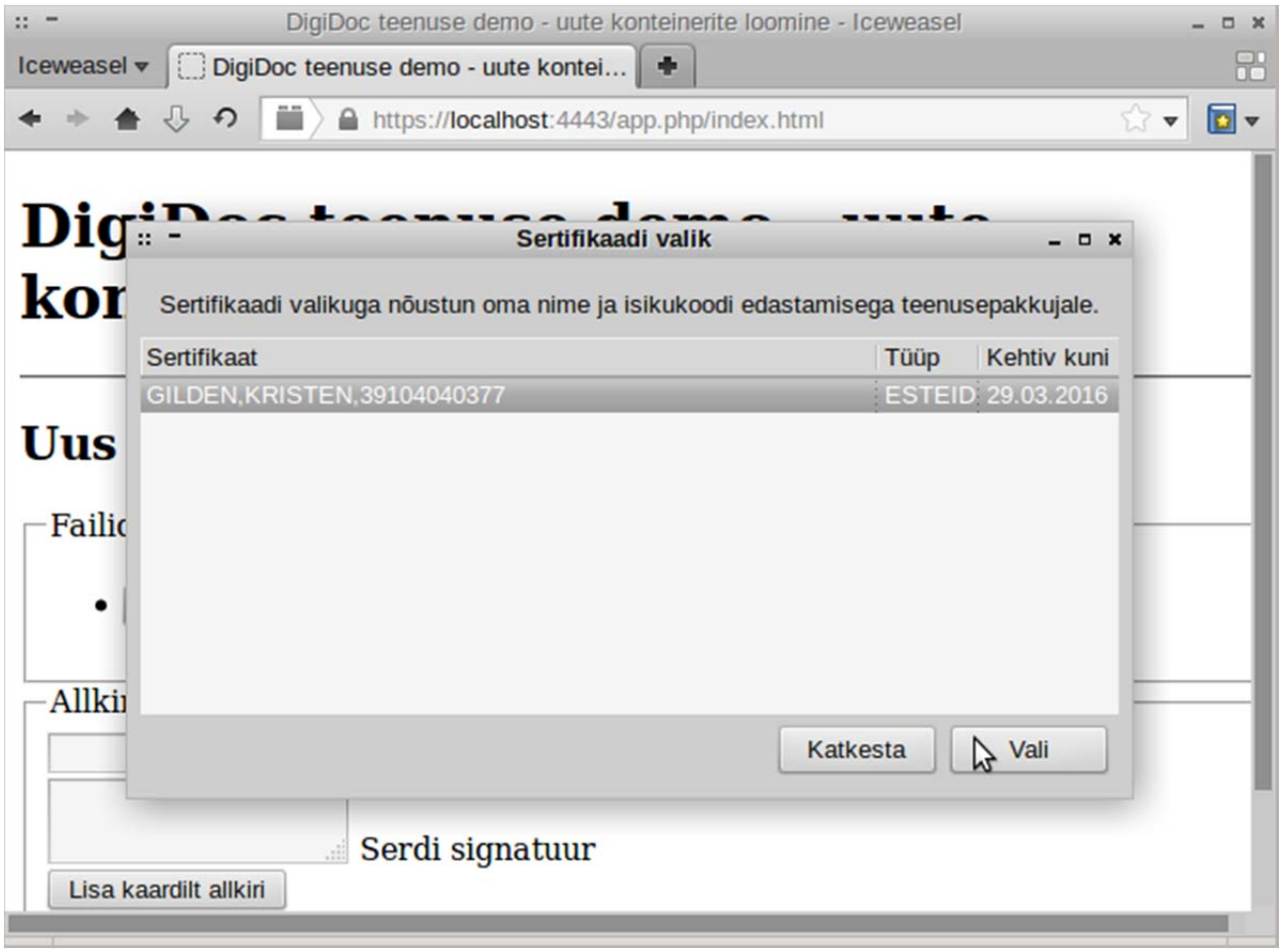
```
<?php
// võrguserveris

use KG\DigiDoc\Signature;

// Tagastab \KG\DigiDoc\Container tüüpi objekti.
$container = $api->create();

// Lisame faili kaustas /home/example. Liidese seisukohast ei ole tähtsust,
// kust fail pärineb - näiteks võis kasutaja eelnevalt oma faili üles laadida
// või rakendus genereeris automaatselt PDF-i.
$container->addFile('/home/example/dokument.txt');
```

Allkirja lisamiseks tuleb teada allkirjastaja serti ning kaardi seerianumbrit. Selleks tuleb suhelda lõppkasutaja lehitsejas asuva pluginaga. Et erinevates lehitsejates on plugin veidi isemoodi, tasub

kasutada olemasolevat *JavaScript*-i teeki, mis erinevused abstraheerib [17]. Lisanud mainitud teegi võrulehele, saab järgneva *JavaScript*-i näite abil allkirjasta sert ja ID-kaardi seerianumber välja lugeda.

```
// lehitsejas

loadSigningPlugin('et');
var plugin = new digidocPluginHandler('et');
var cert   = plugin.getCertificate();
// Muutjas cert on järnevad parameetrid:
//  - id: seerianumber;
//  - cert: allkirjastamise sertifikaat;
//  - CN: allkirjastaja andmed (nimi, perekonnanimi ja isikukood);
//  - issuerCN: serdi väljaandja andmed;
//  - keyUsage: võtme kasutamist kirjeldav parameeter;
```

Kutsudes välja meetodi *getCertificate*, avaneb kasutaja lehitsejas modaalaken, mis palub kasutajal allkirjastamiseks sobiv sertifikaat valida (Joonis 5).



Joonis 5. Serdi valimine lehitsejas

Saadud andmed tuleb allkirja loomiseks tagasi võrguserverile saata ja seal nende abil uus allkirja objekt konteineri külge lisada.

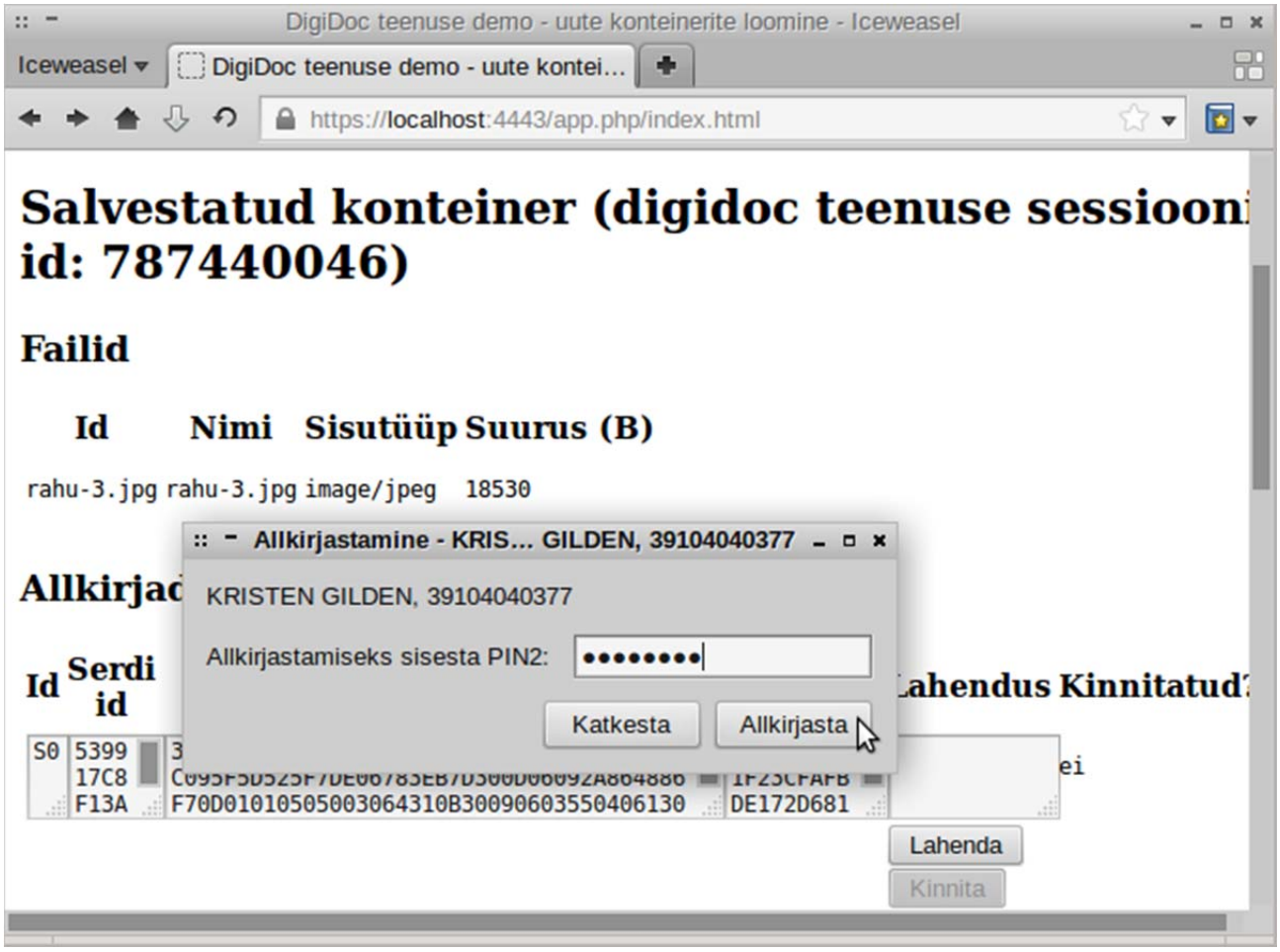
```
<?php
// võrguserveris

// Esimene argument on allkirjastaja sert (lühendatud), teine aga
// kaardi seerianumber (lühendatud).
$signature = new Signature('F1..20', '8F..C0');
$container->addSignature($signature);

// Uuendame DigiDoc võrguteenuses konteineri seisu.
$api->update($container);
```

Nüüd peaks õnnestunud päringu puhul allkirja objektil olema räsi, mis tuleb viimaks allkirjastamiseks taas lõppkasutaja lehitsejale saata.

```
<?php
// võrguserveris

// Allkirja objektil on nüüd räsi, mis tuleb kliendile saata.
$signature->getChallenge();
```

Taas on kasutaja lehitsejas võimalik mainitud *JavaScript*-i teeki kasutada räsi allkirjastamiseks. See avab lehitsejas taas modaalakna, ent seekord tuleb sisestada PIN2 (Joonis 6).

```
// lehitsejas

// 'id' on eelnevalt loetud kaardi seerianumber ja 'challenge' SOAP
// võrguteenuse poolt saadud räsi, mis allkirjastada tuleb.
var solution = plugin.sign(id, challenge);
```



Joonis 6. PIN2 sisestamine allkirja lõplikuks andmiseks.

Saadud tulemus tuleb taas võrguserverile saata ning alles siis on võimalik allkiri lõplikult kinnitada ja kehtivaks muuta.

```
<?php
// võrguserveris

// Eeldame, et räsi allkiri postitati tavalise vormiga lihtsuse mõttes.
$signature->setSolution($_POST['solution']);

// Uuendame taas DigiDoc võrguteenuses konteineri seisu.
$api->update($container);

// Nüüd on konteiner allkirjastatud, salvestame edasiseks töötluseks.
$api->write($container, '/tmp/container.bdoc');

// Teavitame DigiDoc võrguteenust, et antud konteineriga on töö tehtud.
$api->close($container);
```

Kuigi nii antud näide kui ka näiterakendus kasutavad allkirjastamiseks sünkroonsete päringute jada, on probleemile võimalik läheneda ka kasutajasõbralikumalt. Näiteks võib serdi andmed ja allkirjastatud räsi serverile saata asünkroonsete päringutega. Lõppkasutaja jaoks on see oluliselt mugavam viis.

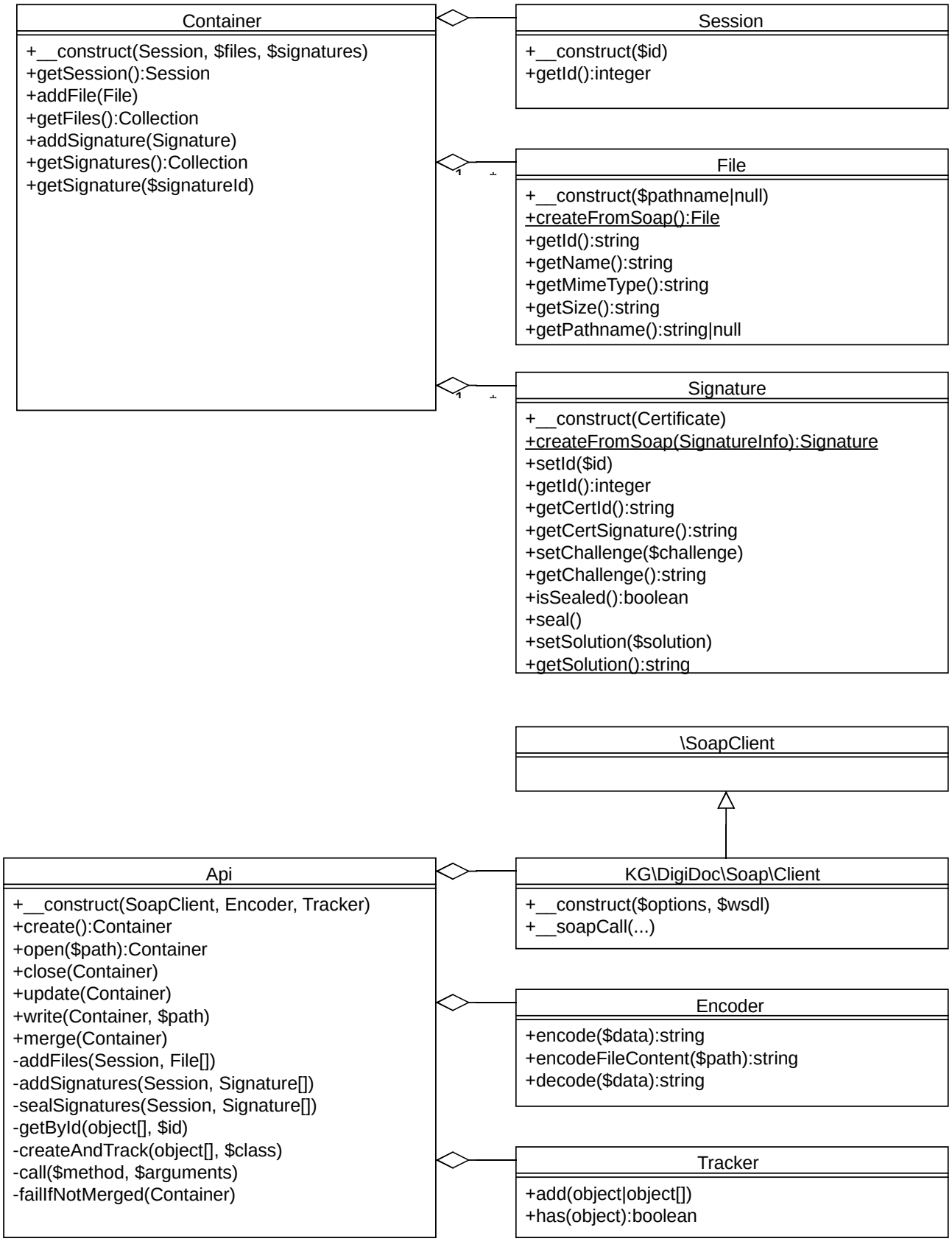
5.1.2. Olemasolevast konteinerist andmete lugemine

Varem loodud konteinerist andmete lugemine sarnaneb uue konteineri loomisele. Tähele tuleks panna, et allkirjastatud dokumendile ei ole tehnilistel võimalik uusi faile lisada – vastasel korral ei klapiks enam eelnevalt antud allkirjad konteineri sisuga.

```
<?php
use KG\DigiDoc\Signature;

// Tagastab \KG\DigiDoc\Container tüüpi objekti.
$container = $api->open('/path/to/container.bdoc');

// Tagastab massiivina kõik konteineris asuvad allkirjad 'Signature'
// objektidena.
$container->getSignatures();

// Tagastab massiivina kõik konteineris asuvad failid 'File' objektidena.
$container->getFiles();

// Lisame uue faili.
$container->addFile('/path/to/file.txt');

try {
    // Kui konteineris on juba mõni allkiri, siis visatakse konteineri
    // uuendamisel erand, sest sel juhul pole uusi faile lubatud lisada.
    $api->update($container);
} catch (ApiException $e) {
    echo $e->getMessage();
}
```

5.2. Arhitektuur

Digiallkirjastamise *PHP* programmeerimisliides *php-digidoc* on täielikult objektorienteeritud, Kontseptuaalselt on see jaotatud kaheks osaks. *DigiDoc* teenusega suhtlev komponent on justkui adapteriks Sertifitseerimiskeskuse võrgurakendusele ja kätkeb endas lihtsaid ja selgeid meetode mudeli muutmiseks. Teine osa – mudel – kujutab endast *DigiDoc* konteinerit, sellega seotud failide ja allkirjade kogumit (Joonis 7).



Joonis 7. php-digidoc klassidiagramm

DigiDoc SOAP (*Simple Object Access Protocol*) teenusega suhtlemiseks on *PHP*-l olemas *SOAP* laiendus, mida antud teek ka kasutab. Lisaks sellele pakub AS Sertifitseerimiskeskus ka teenust

kirjeldavat *WSDL* (*Web Services Description Language*) *XML* dokumenti. Antud dokumendi alusel on automaatselt genereeritud *PHP* objektid, mis vastavad *WSDL* dokumendis defineeritud andmetüüpidele ja võimalikele operatsioonidele. Selle olemasolu vähendab vajadust pöörduda teenuse manuaali poole ja aitab näiteks programmeerimisvigadele kiiremini jälile jõuda.

Kogu *SOAP* teenusega suhtlemine on kapseldatud klassi *\KG\DigiDoc\Api*. Komplekssete käskude asemel on sellel neli lihtsat meetodi. Meetod *Api::create()* tekitab uue konteineri ning seadistab *DigiDoc* teenusega sessiooni. Selle vastand *Api::close()* sulgeb sessiooni pärast suhtluse lõppu. *Api::update()* sünkroniseerib DigiDoc teenuse rakenduses asuva konteineriga – see tähendab lisatakse uued failid, uutele allkirjadele määratakse ülesanded ning lahendatud ülesannetega allkirjad kinnitatakse. *Api::write()* pärib *DigiDoc* teenusest reaalse konteineri faili ning salvestab tagastatud sisu määratud asukohale arvuti kõvakettal.

Konteinerit modelleerib teegis klass *\KG\DigiDoc\Container*. Sinna talletatakse *DigiDoc* teenuse sessiooni id, konteineris asuvad failid ning allkirjad. Et konteineril väliseid sõltuvusi peale eelpool mainitava pole, on seda objekti võimalik sõne kujule ümber viia ja seega ka näiteks sessiooni salvestada. Samuti võimaldab see hõlbust talletamist andmesalve.

Kuna allkirjastamise protsess on küllaltki keerukas tegevus, siis on erilist tähelepanu pööratud kõikvõimalikele eranditele. Kõik vead, mis *DigiDoc* teenus tagastab, tõlgitakse ümber *\KG\DigiDoc\Exception\ApiException* tüüpi eranditeks. Teenuse poolt edastatavatele veakoodidele lisaks pannakse kaasa ka selged kirjeldused. Ühtegi viga ei üritata varjata – nii on kergem nii teegisiseseid kui ka teegi kasutajate rakenduste siseseid probleeme üles leida.

5.3. Paigaldus ja testid

Teegi kasutamiseks teistes projektides tuleb kasutada *Composerit*. Järgnev märge paigaldab teegi versiooni 0.1.0 lokaalselt kausta *vendor/kgilden/php-digidoc*.

```
# Järgnev käsk paigaldab Composeri töökausta nime 'composer.phar' alla.
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar require kgilden/php-digidoc:0.1.0
```

Vältimaks nõutavate failide käsitsi defineerimist igas failis märksõnaga *include*, saab kasutada standardset failide kaasamise komponenti, mille *Composer* ise genereerib ja paigaldab projekti kausta. Et teegi kõik klassid asuvad failisüsteemis sama hierarhiaga, mis nende nimeväljaks deklareeritud (näiteks klass *\KG\DigiDoc\Container* asub failis */KG/DigiDoc/Container.php*), siis on objekti esmakordsel kasutamisel automaatselt võimalik see fail sisse laadida.

```
<?php
require 'vendor/autoload.php';
$api = new \KG\DigiDoc\Api(new \KG\DigiDoc\Client());
```

Kui eelneva käivitamisel ühtegi veateadet ei esinenud, ongi teek edukalt paigaldatud.

5.3.1. Arendusversiooni paigaldus

Ka teeki *php-digidoc* majutatakse *Github'i* võrgurakenduses. Projekti kloonimiseks lokaalsesse arvutisse tuleb jooksutada järgmist käsku.

```
$ git clone https://github.com/kgilden/php-digidoc
```

Sõltuvuste paigaldamiseks saab taas kasutada *Composer'it*. Selleks tuleb projekti juurkaustas jooksutada järgnevad käsud.

```
# Järgnev käsk paigaldab Composeri töökausta nime 'composer.phar' alla.
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar install
```

5.3.2. Testide jooksutamine

Regressioonide vältimiseks on teegile lisatud automaatsed testid. See annab kohest tagasisidet tehtud muudatuse kohta ja võimaldab teeki muuta kartmata, et süsteemi mõni osa lakkaks seetõttu töötamast. Testide jooksutamiseks on kasutatud laialt levinud *PHPUnit* testimise karkassi. Pärast liidese kloonimist saab teste jooksutada projekti juurkaustast järgneva käsuga.

```
$ vendor/bin/phpunit
```

Teegi töökindluse tagamiseks jooksutatakse teste automaatselt pärast igat hoidla uuendamist. Kuna teste jooksutatakse erinevate *PHP* versioonidega, on võimalik veenduda, et teek on ühilduv ka selliste *PHP* versioonidega, mida arendaja arvutisse paigaldatud ei ole.

5.4. Puudused

Kindlasti ei ole valminud liides veel kasutuskõlblik reaalsetes rakendustes. Hetkel on toetatud vaid *BDOC* failivorming ja seega pole võimalik liidesega vanemaid dokumente muuta. Allkirja lisamise sammud on veel kohmakad ega kergesti jälgitavad: seda eriti juhul, kui liidest kasutada mõnes mittetriviaalses rakenduses.

Suurimaks probleemiks on siiski sõltuvus *DigiDoc* võrguteenusest. See on täiendavaks riskiallikaks, sest kasutajal puudub tegelikult kontroll teenuse üle. Et *BDOC* failivormingu

allkirjastamisel toetakse räsi saatmise asemel terviklike failide saatmist, on see täiendav turvarisk, millega rakendused arvestama peaksid. Lisaks sellele on *DigiDoc* teenus tasuline.

Järeldused

Bakalaureusetöö raames leiti, et võrgurakendustes digiallkirjastamise kiire ja lihtsa juurutamiseni on veel pikk maa minna. Korralike terviklahenduste arendamiseks tuleb olla spetsialist suurel hulgal erinevates tehnoloogiates, krüptograafia alustest süsteemi-programmeerimiseni, lõpetades võrkrakendustega.

Kindlasti tuleb mainida, et teistele programmeerijatele mõeldud teegi loomine ja selle arusaadavaks tegemine on keeruline ülesanne. Kuigi allkirjastamiseks loodud liidese arendamise üks eesmärkidest oli see teha võimalikult kergesti kasutatavaks, on loodud tulemuses siiski mõningad puudused. Valminud lõpplahendus läbis arenduse käigus mitu suuremahulist arhitektuurilist muudatust.

Edasisi töösuundi on kahtlemata mitmeid. Esiteks tuleks alustatud liides arendada tasemeni, et see oleks kasutatav ka reaalsetes rakendustes. Lisaks puuduvale funktsionaalsusele tuleb ilmselt loodud koodi refaktoreerida nii, et see oleks paindlikum ja lihtsamini kasutatav. Seda individuaalselt teha pole kindlasti mõistlik ja nii on liides avalikult võrgust avalikult kättesaadav.

Teiseks tuleks üle vaadata olemasolevad ID-kaardi allkirjastamise teegid ja tõsiselt kaaluda nende refaktoreerimist. Et antud hetkel on teekide lähtekood küll avalikult kättesaadav, pole siiski võimalik eraisikutel panustada teekide arendusse. Seega on mõistlik ehk alustada uue C teegi arendusega avalikult ja läbipaistavalt eraldiseisva projektina.

Kokkuvõte

Käesolevas töös uuriti lähemalt Eesti ID-kaardi abil allkirjastamise võimalusi võrgurakendustes. Tutvustati allkirjastamise krüptograafilisi aluseid ning anti ülevaade ID-kaardiga allkirjastamise erinevatest võimalustest.

Valmis uus programmeerimisliides võrgurakendustes allkirjastamiseks *PHP* skriptikeeles. See on avatud lähtekoodiga ning saadaval internetis aadressil https://github.com/kgilden/php-digidoc/. Liides kasutab AS Sertifitseerimiskeskuse poolt pakutavat allkirjastamise võrguteenust *DigiDoc Service*, mis paraku on tasuline. Seetõttu näeb projekti edasine plaan järk-järgulist eemaldumist mainitud teenusest, et allkirjastamine oleks kõigile lihtne ja kättesaadav.

Lisaks programmeerimisliidesele arendati ka selle kasutamist näitav demorakendus, mis on samuti avatud lähtekoodiga ja kättesaadav aadressil https://github.com/kgilden/digidoc-demo. See on ehitatud *Silex*-i mikrokarkassile, kusjuures hoolimata allkirjastamise keerukusest on püütud rakenduse arhitektuur hoida võimalikult lihtsana.

Eestile ainulaadne ID-kaart ja sellega seotud taristu loob eeldused äärmiselt huvitavate digilahenduste loomiseks. Loodetavasti aitab tehtud töö kaasa ID-kaartide süsteemi veelgi laiemale kasutusvõtule võrgurakendustes.

Viited

1. Tammet, T., Mehide, I. (2014). Keerukam näiterakendus digiallkirjastatud avalduste koostamiseks ja töötlemiseks. [WWW] http://eid.eesti.ee/index.php/N%C3%A4iterakendused_keerukam (10.05.2014).

2. AS Sertifitseerimiskeskus. (2013). CDigiDoc Programmer's Guide. [WWW] http://www.id.ee/public/SK-CDD-PRG-GUIDE.pdf (2014).

3. AS Sertifitseerimiskeskus. DigiDoc Service. [WWW] http://www.sk.ee/teenused/kehtivuskinnituse-teenus/digidoc-veebiteenus/ (10.05.2014).

4. World Wide Web Consortium. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). [WWW] http://www.w3.org/TR/soap12-part1/ (10.05.2014).

5. Goldwasser, S., Micali, S., Rivest, R. L. (1988). A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. – *SIAM Journal on Computing*, 17 (2), 281-308. [*Online*] Society for Industrial and Applied Mathematics (11.05.2014).

6. Diffie, W., Hellman, M. (1976). New directions in cryptography. – *IEEE Transactions on Information Technology*. 22 (6), 644-654. [*Online*] IEEE Xplore (03.05.2014).

7. Rivest, R. L., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatuures and publik-key cryptosystems. – *Communications of the ACM*, 21 (2), 120-126. [*Online*] ACM Digital Library (04.05.2014).

8. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R. (2007). OpenPGP Message Format. [WWW] https://tools.ietf.org/html/rfc4880 (03.05.2014).

9. National Institute of Standards and Technology. (2013). Digital Signature Standard. [WWW] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf (10.05.2014).

10. Housley, R., Ford, W., Polk, W., Solo, D. (1999). Internet X.509 Public Key Infrastructure. [WWW] http://www.ietf.org/rfc/rfc2459.txt (19.05.2014).

11. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C. (1999). X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP. [WWW] http://www.ietf.org/rfc/rfc2560.txt (18.05.2014).

12. Trüb Baltic AS. (2012). ID-card specification. [WWW] http://www.id.ee/public/TB-SPEC-EstEID-Chip-App-v3.4.pdf (11.05.2014).

13. World Wide Web Consortium. (2003). XML Advanced Electronic Signatures (XAdES). [WWW] http://www.w3.org/TR/XAdES (17.05.2014).

14. AS Sertifitseerimiskeskus. (2004). DigiDoc formaadi kirjeldus. [WWW] http://www.id.ee/public/DigiDoci_vorming_1.3.2.pdf (17.05.2014).

15. AS Sertifitseerimiskeskus. (2013). BDOC – digitaalallkirja vorming. [WWW] http://id.ee/public/bdoc-spec21-est.pdf (18.05.2014).

16. Leff, A., Rayfield, J. T. (2001). Web-Application Development Using the Model/View/Controller Design Pattern. – *Enterprise Distributed Object Computing Conference, EDOC 2001, Seattle, Washington, September 04-07: Proceedings. – Fifth IEEE International*, 118-127. [*Online*] IEEE (19.05.2014).

17. AS Sertifitseerimiskeskus. (2013). JavaScripti klienditeek (idCard.js). [WWW] http://id.ee/index.php?id=30369 (30.05.2014).

Lisa 1 – ID-kaardi allkirjastamise sertifikaadi näidis

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1f:fb:ff:ae:ac:09:5f:5d:52:5f:7d:e0:67:83:eb:7d
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=EE, O=AS Sertifitseerimiskeskus, CN=ESTEID-SK 2011/emailAddress=pki@sk.ee
        Validity
            Not Before: Oct 17 06:04:16 2013 GMT
            Not After : Mar 29 21:00:00 2016 GMT
        Subject: C=EE, O=ESTEID, OU=digital signature, CN=GILDEN,KRISTEN,39104040377, SN=GILDEN,
GN=KRISTEN/serialNumber=39104040377
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    # 257 baiti kujul 00:01:..:FF
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage: critical
                Non Repudiation
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.10015.1.1.3.3
                  User Notice:
                    Explicit Text: none
                  CPS: http://www.sk.ee/cps/

            X509v3 Subject Key Identifier:
                4A:78:8A:F4:19:DC:AC:74:99:BC:F8:29:19:0F:25:85:1B:CB:04:EA
            qcStatements:
                0.0......F..0......F..
            X509v3 Authority Key Identifier:
                keyid:7B:6A:F2:55:50:5C:B8:D9:7A:08:87:41:AE:FA:A2:2B:3D:5B:57:76

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://www.sk.ee/repository/crls/esteid2011.crl

    Signature Algorithm: sha1WithRSAEncryption
         # 256 baiti kujul 00:01:..:FF
```

Lisa 2 – DDOC ja DDOC failivormingu näited

Mõlema vormingu puhul on allkirjastatud kaks faili vastava sisuga.

```
helloworld.txt: Hello, world!
foobar.txt:     Foo Bar
```

Näitamaks, kuidas näeb välja konteiner erinevates vormingutes mitme allkirjaga, on nii DDOC kui ka BDOC vormingu puhul antud kaks allkirja.

Näited asuvad CD-plaadil kaustas *lisa_2_ddoc_bdoc_vormingu_naited*.

Lisa 3 – lähtekood

Lähtekood asub CD-plaadil kahes kaustas. Võrgurakendus asub kaustas *lisa_3_lahtekood_rakendus*. Rakenduse jaoks arendatud programmeerimisliides asub kaustas *lisa_3_lahtekood_api*.

Lisa 4 – DigiDoc C teegi kasutamise näidiskood

DigiDoc C teegi analüüsimiseks kasutati järgnevat koodi. Kompileeritud programmiga samas kaustas asus fail *hello.txt* sisuga *„Hello, world!“*. Sellest genereeris programm DDOC vormingus faili *„hello.ddoc“*.

```
/**
 * A very basic hello world example of using libdigidoc. It creates a
 * DigiDoc v1.3 file container from "hello.txt" sitting in the same
 * directory.
 * @author Kristen Gilden <kristen.gilden@gmail.com>
 */
#include <DigiDocLib.h>
#include <DigiDocObj.h>
#include <DigiDocDefs.h>
#include <DigiDocGen.h>

int main() {
    SignedDoc *container = NULL;  // The new file we're about to create
    DataFile  *file      = NULL;  // The file to be added to the new container

    // DigiDoc initialization. OpenSSL initialization is most likely the most
    // important part of it.
    initDigiDocLib();

    // Allocates memory for creating the new container. Unfortunately there's
    // no support for BDOC.
    SignedDoc_new(
        &container,
        DIGIDOC_XML_1_1_NAME,
        DIGIDOC_XML_1_3_VER
    );

    // Allocates memory for the structure representing the file to be added
    // to the container and initializes it.
    DataFile_new(
        &file,              // where the memory should be allocated
        container,          // current container
        NULL,               // file id
        "hello.txt",        // path to the file to be included
        "EMBEDDED_BASE64",  // embedding type
        "text/plain",       // file's mime type
        14,                 // file size (hardcoded)
        NULL,               // digest of file contents (for separate files)
        0,                  // digest length (for separate files)
        NULL,               // digest type (e.g SHA1)
        "UTF-8"             // file's character set
    );

    // Generates the actual signed document to filesystem named "hello.ddoc".
    createSignedDoc(container, NULL, "hello.ddoc");

    return 0;
}
```

Programm kompileeriti 64-bitise GNU/Linux distributsioonil käsuga, kusjuures eelnevalt oli paigaldatud teek *libdigidoc* ja kõik selle sõltuvused.

```
gcc \
  -Wall \
  -Ilibdigidoc/libdigidoc \
  -Ilibdigidoc \
  -L./usr/lib/x86_64-linux-gnu/ \
  -l :libdigidoc.so.2 \
  dd_create.c -o dd_create
```

Faili „*hello.txt*“ sisu oli

```
Hello, world!
```

Programm genereeris järgneva DigiDoc faili (vormindus on parendatud lugemise kergendamiseks)

```
<?xml version="1.0" encoding="UTF-8"?>
<SignedDoc
  format="DIGIDOC-XML"
  version="1.3"
  xmlns="http://www.sk.ee/DigiDoc/v1.3.0#"
>
  <DataFile
    ContentType="EMBEDDED_BASE64"
    Filename="hello.txt"
    Id="D0"
    MimeType="text/plain"
    Size="14"
    xmlns="http://www.sk.ee/DigiDoc/v1.3.0#"
  >
    SGVsbG8sIHdvcmxkIQ0K
  </DataFile>
</SignedDoc>
```

Kogu lähtekood ja saadud tulemus on CD-plaadil kaustas *lisa_4_digidoc_c_teegi_kasutamine*.