TALLINNA POLÜTEHNIKUM

IT ja telekommunikatsioon erialaosakond

Keijo Kapp

Kõrge terviklusega andmeid talletava andmebaasilahenduse prototüüp

Lõputöö

ARVUTID JA ARVUTIVÄRGiD

PA-12A

Juhendajad: Deivis Treier, Tarmo Teder

Tallinn 2015

AUTORIDEKLARATSIOON

Deklareerin, et käesolev lõputöö, mis on minu iseseisva töö tulemus, on esitatud Tallinna Polütehnikumi lõputunnistuse taotlemiseks Arvutid ja arvutivõrgud erialal. Lõputöö alusel ei ole varem eriala lõputunnistust taotletud.

Autor Keijo Kapp ….....

(allkiri ja kuupäev)

Töö vastab kehtivatele nõuetele.

Juhendaja Deivis Treier .....

(allkiri ja kuupäev)

Juhendaja Tarmo Teder .....

(allkiri ja kuupäev)

Sisukord

Lühendid ja terminid

JSON – *JavaScript Object Notation* – standard andmete hoidmiseks ja edastamiseks tekstikujul

JSONB – *JSON Binary* – binaarne struktueeritud JSON andmetüüp

ISKE – Peamiselt Eesti avaliku sektori infovaradel kasutatav kolmeastmeline etalonturbe süsteem

SK – Sertifitseerimiskeskus AS – peamiselt krüptograafiliste võtmete ja allkirjade sertifitseerimisteenust pakkuv ettevõte.

MVC – *Model-View-Controller* – konventsioon, milles eristaatkse andmeid (*model*), vaadet/kasutajaliidest (*view*), kasutajaliidese loogikat (*controller*) ja enamasti ka äriloogikat (*service*)

HSM – *Hardware Security Module* – riistvaraline krüptoseade, sarnane ID-kaardile

Sissejuhatus

ISKE on Eestis kasutatav kolmeastmeline infosüsteem, mida kasutatakse peamiselt avaliku sektori, riigi ja kohalike omavalitsuste, infosüsteemide turvalisuse tagamisel. (RIA, 2014) See defineerib muuhulgas meetmed, mille eesmärk peab olema saavutatud kas siis meedet rakendades või mõnel teisel viisil.

Infosüsteemide turvameetmete süsteemi ISKE järgi tähendab andmete terviklus andmete õigsuse/täielikkuse/ajakohasuse tagatust, autentsust ning volitamatute muutuste puudumist (RIA, 2014). Andmete terviklust saab tagada vastavaid meetmeid rakendades

Lõputöö eesmärk on

….....

Prototüüp (ingl.k. *Proof Of Concept*) eesmärk on näidata kontseptsiooni reaalset toimimist. Lõputöö kontseptsioon koosneb 4 osast:

- Andmebaasi kirjete krüptoaheldamine
- Andmebaasis olevate andmete digiallkirjastamine
- Kirjete ajaloo hoidmine
- JSONB andmetüübi kasutamine andmete hoidmiseks

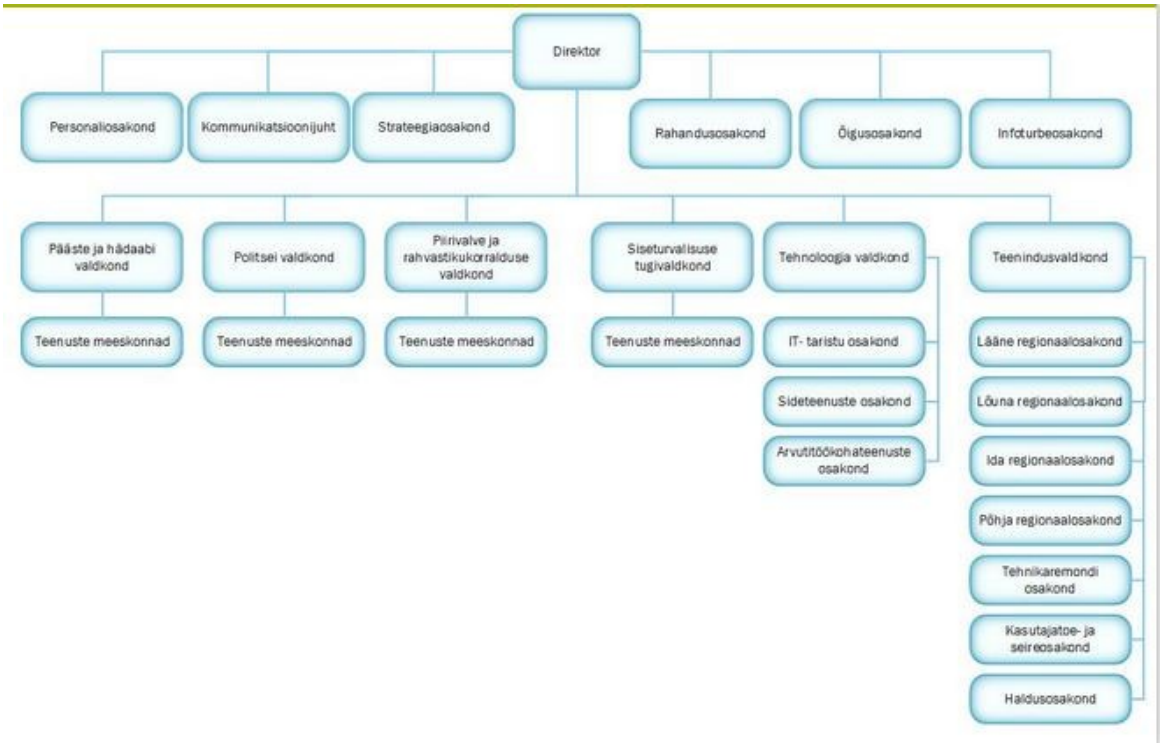
Lõputöö käsitleb ainult kolme esimest kontseptsiooni osa. Neljas on siiski kontseptsiooni töötamiseks vajalik, sest see aitab kergemini tagada andmete terviklust.

1. Ettevõttest

Siseministeeriumi Infotehnoloogia- ja Arenduskeskus on Eesti Siseministeeriumi haldusalas IT teenuseid pakkuv ettevõte. Ettevõte arendab ja haldab muuhulgas infosüsteeme ning asutuste töökohti.

Ettevõtte kontorid asuvad Tallinnas, Tartus, Pärnus ja Jõhvis. Töötajaid on ettevõttel umbes 250. Peamised kliendid on Politsei- ja Piirivalveamet, Päästeamet, Häirekeskus ja Siseministeerium. Lisaks haldab ettevõte ka riigi operatiivraadioside võrku TETRA. Tehniline partnerlus ja tootetugi on Maanteeametiga.

Ettevõtte struktuur on kirjeldatud järgneval skeemil. Lõputöö autor osales piirivalve ja rahvastikukorralduse valdkonna töös.



2. Lahenduse eesmärk

Ettevõttel on vaja luua tehniline võimekus kõrge terviklusega andmete hoidmiseks rakendades vastavaid tarkvaralisi turvameetmeid. Lahendus peaks olema võimalikult abstraktne, et seda saaks kasutada erinevate infosüsteemide loomiseks.

Lahenduse loomise hõlmab erinevate võimaluste uurimist ja olulisimate realiseerimist. Kuna lahendus ei saa olla suunatud kindlale infosüsteemile, ei saa ka rääkida konkreetse funktsionaalsuse/koodi/teegi kirjutamisest.

(TODO: järgmine lõik tuleks eemaldada)

Aluseks on võetud Eesti Isikut Tõendavate Dokumentide Andmeregistri nõuded. Lõputöö autor osaleb ise selle arendamisprotsessis ja tegeleb vähemalt osaliselt kõigi küsimustega, mis puudutavad andmete salvestamist, allkirjastamist, lugemist ja andmete muutuste jälgitavust.

2.1. Andmebaasi kirjete krüptoaheldamine

Eesmärk on vastavalt ISKE meetmele **HT.10 Andmebaasi kannete krüptoaheldamine** (RIA, 2014) luua funktsionaalsus, mis seob andmebaasitabeli piires kirjed selliselt, et igasugused, nii tahtlikud kui rikke tõttu tekkinud, manipulatitsioonid andmetega oleksid leitavad. Andmed seotakse ridade väärtusest tuletatava räsiväärtuse kaudu nii, et see räsi oleks seotud ka eelmise rea räsi väärtusega.

Sidumiseks kasutatakse räsialgoritmi, mis vastab **HT.52 Lisanõuded krüptovahenditele** meetmele. See välistab praeguseks ebaturvaliseks tunnistatud MD räsialgoritmide perekonna. (RIA, 2014).

Kuigi ISKE meede seda välja ei too, ei ole lihtsa aheldamisega võimalik tagada andmebaasis oleva viimase kirje muutumatust, sest sellest ei sõltu mõne teise kirje räsi. Samuti on võimalik märkamatult muuta viimaseid kirjeid, kui muudetavast kirjest alates ahel üle arvutada. Selle vältimiseks peab aheldamist teostav funktsionaalsus võimaldama viimase kirje räsi turvalist salvestamist. Kontseptsiooni järgi salvestatakse räsi mõnes teises süsteemis, kuhu infosüsteemi meeskonnaliikmetel ligipääsu ei ole.

Meetme eesmärgi saavutamisel ei piisa ainult krüptoaheldamisest - seda ahelat peab ka perioodiliselt kontrollima. Kontrollmehhanismi loomine ja rakendamine ei ole lõputöö skoobis.

2.2. Andmete digiallkirjastamine

Digiallkirjastamine toimub vastavalt ISKE **HT.34 Digiallkirja kasutamine** meetmele. Meede lubab allkirjastamiseks kasutada ainult mehhanisme, mis vastavad Eesti digitaalallkirja seadusele. (RIA, 2014) Lisaks terviklikkuse tagamisele annab see andmetele vähemalt Eesti piires ka juriidilise tõendusväärtuse.

Digiallkirjastamisele kuuluvad kõik kõrge terviklikkuse nõudega andmed, mida potensiaalselt väljastatakse infosüsteemist. See tagab ka vähem turvalistes keskkondades andmete terviklikkuse ja autentsuse. Infosüsteemi sees on andmete terviklikkus tagatud krüptoaheldamise meetmega ning kõigi andmete allkirjastamine on nii rahalistel kui tehnilistel põhjustel ebaotstarbekas.

Kontseptsiooni järgi on nendeks andmeteks dokumendid (nt. avaldused, taotlused) koos nende juurde käivate lisadega (nt. pilt, allkiri). Masintöödeldavad dokumendid on JSON vormingus ning vajadusel käib nende juurde lisana ka tavainimesele loetavas HTML vormingus dokument.

2.2.1. JSON vorming

ECMA-404 ehk JSON (Javascript Object Notation) on viimastel aastatel väga populaarseks saanud vorming masintöödeltavate andmete hoidmiseks ja edastamiseks tekstikujul. Seda kasutatakse laialdaselt veebis, andmebaasides, konfiguratisioonides ja paljudel teistel eesmärkidel.

Vorming baseerub väikesel osal *ECMA-262 3rd Edition* standardist, mida tuntakse programmeerimiskeelena *Javascript*. (ECMA International, 2013) JSON-i käsitletakse

paljudel juhtudel XML-i kaasaegse anternatiivina, olles palju mahuefektiivsem ja enamikel juhtudel ka inimesele loetavam. Lisaks on JSON'il erinevalt XML'ist tugi erinevate andmetüüpidele (massiiv, number, string, objekt jne.) analoogselt Javascript'iga.

JSON andmekomplekte nimetatakse ka JSON dokumentideks. Andmed on dokumendis hierarhilise puuna, kusjuures originaalstandardi (RFC4627) järgi võib puu juurelement olla ainult objekt või massiiv. ECMA-404 standardile vastavad implementatsioonid võimaldavad juurelemendina ka teisi andmetüüpe kasutada, kuid mitte kõik kasutavad süsteemid (sh. lõputöös kasutatav andmebaasitarkvara PostgreSQL) ei toeta seda, mistõttu on mõistlik sellest hoiduda. Juurelemendina objekti kasutamine välistab ka teatud XSS (ingl.k. *Cross-site scripting*) turvaprobleemi, sest selliselt vormindatud dokument ei ole terviklik Javascript'i programm, kuid see-eest kõigi parserite poolt aksepteeritav JSON dokument.

JSON dokument võib välja näha näiteks selline:

```
{
      "Image": {
            "Width":  800,
            "Height": 600,
            "Title":  "View from 15th Floor",
            "Thumbnail": {
                  "Url": "http://www.example.com/image/481989943",
                  "Height": 125,
                  "Width":  "100"
            },
            "IDs": [116, 943, 234, 38793]
      }
}
```

(Crockford, 2006)

2.2.2. JSONB andmetüüp

PostgreSQL 9.4 tõi uuendusena kaasa JSONB andmetüübi, millega saab hoida JSON dokumenti andmebaasis strukteeritult binaarsel kujul. See võimaldab käsitleda üksikuid JSON dokumendi välju tabeli väljadena. Näiteks saab JSONB andmetüübiga üksikuid JSON välju lugeda, kirjutada ja indekseerida. Selline dünaamilisus võib radikaalselt muuta andmete hoidmise põhimõtteid. Näiteks kaob vajadus paljude relatsioonide

(võõrvõtmete) kasutamise järeldusele, sest andmed saab kirjutada otse JSON dokumendi sisse.

2.3. Kirjete muudatuste ajaloo hoidmine

Kõrge terviklus eeldab ka seda, et andmeid reaalselt ei kustutata ega muudeta, vaid märgitakse mitteaktiivseteks. Kustutamise või muutmise teeb võimatuks ka krüptoaheldamise meede, mis sisuliselt ongi selleks, et selliseid muutmisi või kustutamisi tuvastada. Need manipulatsioonid tuleb teostada rakenduses loogiliselt. Näiteks peab kirje kustutamise asemel lisama uue kirje, mis märgib eelmise (loogiliselt kustutatud) kirje mittekehtivaks.

Vaatamata probleemi esinemisele paljudes kaasaegsetes infosüsteemides ei ole üldist olemasolevat lahendust sellele. Üks enam-kasutatavamaid lahendusi on mitteaktiivsete kirjete kandmine eraldi tabelisse, kuid krüptoaheldamise meetme rakendamisel on see keeruline, sest ahel peab olema katkematu ning ahela lülide hoidmine erinevates tabelites teeb ahela kontrollimise keerukaks.

Kontseptsiooni järgi peab terve tabel olema konsistentne tervik ning sisaldama kogu kirjete ajalugu. Lisaks peab olema võimalik kiiresti leida iga üksiku kirje muutused ajas – nii enne kui peale vastavat kirjet.

3. Lahenduse analüüs

3.1. ISKE

ISKE on Riigi Infosüsteemi Ameti poolt väljatöötatud kolmeastmeline etalonturbe süsteem. Selle aluseks on võetud Saksamaal kasutatav *IT Baseline Protection Manual*. ISKE rakendamise eesmärk on tagada infosüsteemides töödeldavate andmete piisava tasemega turvalisus. Peamiselt on süsteem loodud riigi ja kohalike omavalitsuste andmekogude turvalisuse tagamiseks. (RIA, 2014)

Infosüsteemide vastavuse nõue ISKE turbeastmele määratakse vastava infosüsteemi kohta käiva määrusega. Näiteks Isikut Tõendavate Dokumentidega Andmekogu (ITDAK) turbeklassi määrab "Isikut tõendavate dokumentide andmekogu pidamise põhimäärus".

Meetmeid võib ka mitte rakendada, kui eesmärk suudetakse teisiti tagada. Sellisel juhul peab looma vastavad dokumendid meetme mittetäitmiseks.

Lõputöö eesmärk on luua kahe konkreetse ISKE turvameetme rakendamise prototüübid, mis on eelpool kirjeldatud. Neid meetmeid pole sellisel kujul ettevõttes varem rakendatud.

3.3. Kasutatavad tehnoloogiad

3.3.1 Java

Java on multiplatvormne programmeerimiskeel ja selle käivitusmootor (virtuaalmasin). Javat kasutatakse laialdaselt serveri-, töölaua-, mobiili- jm. rakenduste loomiseks. Java peamiseks eelisteks on paljude platvormide tugi, mis võimaldab peaaegu ilma platvormispetsiifilise koodi kirjutamist ja ülekompileerimist jooksutada programmi

mitmetel platvormidel. Lisaks Java'l väga suur kausutajaskond, mis tagab kerge lahenduse leidmise enamikele probleemidele.

Probleemidena on Java'l võrreldes teiste kõrgtaseme keeltega suhteliselt keeruline sõltuvuste (ingl.k. *dependency*) haldus, mitu täiesti erinevat ehitussüsteemi (Maven, Ant/Ivy) ning liiga palju erinevaid koodikonventsioone, millest halvemal juhul ei peeta kinni. Olulisimaks probleemiks peab autor asünkroonse I/O teostamise keerulisust, millest tuleneb massiivne erinevate lõimede kasutamine, mis omakorda võib tekitada raskesti avastatavaid vigu piisavalt kogenematul arendajal.

3.3.2 Groovy

Groovy on Java platvormil ja süntaksil baseeruv programmeerimiskeel. Groovy loob võimaluse kasutada nn. *duck typing*'ut, mis tähendab, et andmetüüpide eraldi väljatoomine koodis ei ole vajalik. See ja teised Groovy võimalused võimaldavad kirjutada dünaamilist ning kohati ilusamat ja paremini loetavat koodi kui Java.

Teisalt on Groovy harjumatu varasemalt Java arendajale, sest süntaks on küll sarnane, aga ka oluliselt erinev. Võib olla keeruline lugeda koodi, kus ei ole muutujate väärtuste tüübid eraldi välja toodud. Lisaks on Groovy olulselt aeglasem võrreldes Java'ga, sest andmetüüpide määramine ja sellest tulenevalt käivitatava koodi valikute otsustamine toimub programmi käitamise ajal.

3.3.3 Grails

Grails on Groovy programmeerimiskeelel baseeruv MVC raamistik veebirakenduste loomisks. See baseerub Java *Spring Framework* raamistikul. Erinevalt Spring Framework raamistikust on Grails üles ehitatud vastavalt *Conventsion Over Configuration* (COC) arendusmeetodile. See tähendab, et näiteks ei pea arendaja kontrolleri (ing.k. *controller*) koodi märgistama, et tegemist on kontrolleriga ning määrama, millisele tegevusele (*action* või *route*) see vastab. Piisab, kui kontroller on vastavalt konventsioonile nimetatud ja selle kood asub vastavas failis. See võib oluliselt kiirendada produktiivsust, sest arendaja peab füüsiliselt vähem koodi kirjutama. Samas eeldab see kohati väga mahukate konventsioonide teadmist.

Kohati võib Grails'i käsitleda ka omaetteplatvormiga, sest see sisaldab spetsiifilist funktsionaalsust (näiteks teadete logimine, testimine ja dokumentatsioon), ehitus- ja

sõltuvuste haldamise süsteemi ja konventsioone teekide arendamiseks ja rakenduse konfigureerimiseks.

3.3.4 PostgreSQL

PostgreSQL on laialdaselt kasutatav relatsiooniline andmebaasisüsteem. Erinevalt konkureerivatest andmebaasisüstemidest (MySQL, MSSQL) on PostgreSQL'il oluline eesmärk järgida täpselt SQL standardit. PostgreSQL on üks lihtsamini liidestatavaid andmebaasisüsteeme, omades samal ajal spetsiifiilisi võimalusi nagu näiteks JSON ja JSONB andmetüübid.

Lõputöö käsitleb muuhulgas PostgreSQL'i-spetsiifilise andmetüübi JSONB kasutamist, mille tugi tuli PostgreSQL'ile alles üsna hiljutise versiooniga 9.4.

3.3.5 PKCS11 ja HSM

PKCS11 on PKCS (ingl.k. *Public Key Cryptography Standards*) perekonna standard. See käsitleb krüptoseadmega suhtlemist tarkvaratasandil. Sellised seadmed on näiteks ID-kaarid, pangakaardid, HSM'id või ka virtuaalsed seadmed (näiteks PKCS11SPY või SoftHSM). Üldiselt väljastab iga krüptoseadme tootja oma krüptoseadmele PKCS11 standardile vastava teegi, mis laetaks dünaamiliselt kasutaja rakendusse.

PKCS11 defineerib muuhulgas, kuidas kasutada ja manipuleerida krüptoseadmes olevaid võtmeid ja sertifikaate. PKCS11 teekide konfigureerimine toimub tavaliselt rakenduseväliselt näiteks eraldi konfiguratsioonifaili või keskkonnamuutujate kaudu.

4. Töö teostus

4.1. Krüptoaheldamine

Krüptoahealdamise praktilise osa eesmärk on luua olukord, kus ükski osapool ei saaks ainuisikuliselt märkamatult muuta ega kustutada andmebaasis olevaid andmeid. Osapoolteks on kontseptsiooni järgi vähemalt kaks administratiivset tsooni: ühes tsoonis asub rakendus ja andmebaas koos vastavate administraatoritega ja teiste meeskonnaliikmetega ning teiseks tsooniks on mõni teine osakond, kontseptsiooni järgi näiteks infoturbe osakond. Lõputöö ei käsitle olukorda, kus andmete muutmine peab olema absoluutselt välistatud, välja arvatud vastutava isiku, näiteks organisatsiooni juhi, nõusolekul.

Autor on käsitlenud mitut erinevat lähenemist. Mõlemal juhul peab vastav funktsionaalsus laias laastus tegema igal kirje sisestamisel 4 sammu. Need sammud tuleb teostada vahetult enne kirje sisestamist ning atomaarselt.

1. Eelmise kirje räsi laadimine vahemälust, andmebaasist või muust allikast
2. Sellest räsist ja sisestatava rea väärtusest uue räsi arvutamine
3. Uue räsi toimetamine teise administratiivsesse tsooni koos toetavate andmetega (nt. aeg, rea identifikaator, eelmine räsi)
4. Sisestatava rea muutmine kirjutades saadud räsi vastavale väljale.

4.1.1. Syslog

Räsi saatmiseks serverisse otsustati kasutada *syslog* nimelist protokolli. Tegemist on protokolliga, mida kasutatakse keskseks ja standardseks logimiseks. Syslog'i kasutamine on tavaliselt väga lihtne ning on dokumenteeritud “The GNU C Library”

dokumentatsioonis. (Free Software Foundation, 2015) Tihti käsitletaksee *syslog* protokolli ebaturvalise ning ebastabiilsena. Võib esineda logiteadete kadumist ning halvimal juhul on võimalik teateid manipuleerida. Siiski on võimalik neid probleeme vältida kasutades vastavaid meetmeid, näiteks IP/TCP ja TSL (ingl.k. *Transport Level Security* – võrguliikluse krüpteerimise mehhanism).

4.1.1. Grails'i ORM-i kasutamine

(TODO: vb ei olegi mõistlik sellest siin jahvatada, see mingi lamp mõte, mida ma isekeskis arutasin)

Üks lähenemisviis oleks teha need sammud rakenduse sees kasutades Grails raamistiku ORM-i (GORM) funktsionaalsust. Nimelt saab GORM-i objektidele (domeeniobjektid) lisada meetodi, mis käivitatakse iga kord vastavata tegevusega seoses. Antud juhul oleks selleks meetodiks *beforeInsert*.

Selline lähenemine oleks turvalisuse seisukohalt kõige loogilisem, sest rakendus peaks ainsana andmebaasi kirjeid sisestama ning seega saaks vastutada selle eest, et räsi vastab sellele, mida rakendus andmebaasiserverile saadab. Seda saab rakendus teha ise kirjeid krüptograafiliselt aheldades kasutades eelpool nimetatud meetodit.

Mitmetel põhjustel aga selline lahendus antud kontseptsiooni järgi ei töötaks. Mõned neist on loetletud ka siin:

- Sellisel juhul oleks väga raske tagada atomaarsust. Arvestades, et rakendusest luuakse mitu instantsi, on ainuke võimalus kasutada andmebaasi lukustamist, mis ei ole piisavalt suure koormuse juures aksepteeritav.
- Eelmise räsi laadimine on keerukas. Kui laadida räsi iga kord andmebaasist, tekitaks see jõudluse probleeme. Vahemälus hoidmine jälle võib viia mittekonsistentsete olukordadeni erinevate rakenduse instantside suhtes.
- Autor ei ole piisavalt kogenud, et usaldusväärselt tagada lõimedevaheline võistlussituatsioonide ärahoidmine. Väikseimgi hooletus võib tekitada raskesti tuvastatavaid vigu.

4.1.2. Andmebaasi päästikute kasutamine

Andmebaasi päästik (ingl.k. *trigger*) on funktsioon, mis käivitatakse andmebaasimootoris kasutaja poolt defineeritud sündmusel või sündmustel. Andmebaaside termonoloogias kutsutakse kasutaja poolt defineeritud funktsioone terminiga *Stored Procedure*. Neid funktsioone saab kasutada ka teistel eesmärkidel, nt. kasutaja poolt defineeritud andmetüübid, matemaatilised funktsioonid, indekseerimine jne.

PostgreSQL päästikute kirjutamine ja kasutamine on suhteliselt hästi dokumenteeritud. Päästikuid saab kirjutada erinevates programmeerimiskeeltes: lihtsamatel juhtudel pgSQL, keerulistemal PL/pgSQL, PL/Python või PL/v8 (Javascript) ning kõige keerulistemal juhtudel C, C++ või muu keel, mida C-keeles kirjutatud andmebaasimootor saab linkida. (The PostgreSQL Global Development Group, 2015; Hitoshi Harada, 2015)

Kuna vajalik funktsionaalsus on suhteliselt keeruline ning PL (ingl.k. *Procedural Language*) eelliidestega keelte funktsionaalsus on piiratud (peamiselt turvalisuse ja stabiilsuse tagamiseks), otsustas autor kasutada C keelt. Autoril on varasemalt suhteliselt pikk kogemus C ja teiste madala taseme programmeerimiskeelte kasutamisel.

4.1.3. Päästiku realiseerimine C-keeles

Krüptoaheldamise päästiku lähtekood koosneb viiest lähtekoodi failist, millele lisandub *Makefile* ja *README.md*. *Makefile* on *GNU Make* nimelise ehitustööriista poolt kasutatav fail, *README.md* on *Markdown* formaadis väike dokumentatsioonifail.

Päästiku kompileerimiseks saab kasutada *GNU Make* tööriist. Kompileerimiseks peab lähtekoodi juurkaustas käivitama käsu

```
make
```

Programm *make* loeb vastavas kaustas olevat faili *Makefile* ning käivitab selles failis oleva kompileerimise käsu. Kompileerimise väljundiks objektifail *hashchain-trigger.so*, mis ekspordib andmebaasiserverile kaks funktsiooni:

- *hashchainTrigger* – funktsioon, mis räsiaheldab sisestatavad kirjed
- *logHashTrigger* – funktsioon, mis saada räsiväärtuse *syslog*'i serverile

Stored Procedure registeerimiseks peab vastavas andmebaasis käivitama SQL käsu

```
CREATE FUNCTION "hashchainTrigger"()        -- funktsiooni nimeks on 'hashchainTriiger'
       RETURNS trigger                      -- funktsiooni kasutatakse päästiku jaoks
       AS '/path/to/hashchainTrigger.so'    -- kompileerimise väljundfail
       LANGUAGE C;                          -- funktsioon on kirjutatud C keeles
```

Päästiku loomiseks peab käivitama käsu

```
CREATE TRIGGER hashchain BEFORE INSERT ON tabeli_nimi
       FOR EACH ROW EXECUTE PROCEDURE "hashchainTrigger"();
```

See loob päästiku, mis käivitatakse igal rea sisestamisel **enne** sisestamist. Kasutades ainult räside logimise funktsionaalsust, peab veenduma, et trigger käivitatakse pärast räsiaheldamist teaostavat trigger.

Kogu päästiku kood on toodud lisas. (*vt. Lisa X*

4.1.4. PL/pgSQL kasutamine päästikutes

Arusaadavatel põhjustel väldib ettevõte madalataseme keelte nagu C kasutamist. Põhjuseks tuuakse näiteks see, et selliselt kirjutatud kood ei ole tihtipeale platvormist sõltumatu (nt. erinevad andmetüüpide suurused).

Osa päästiku funktsionaalsuse jaoks on võimalik kasutada PL/pgSQL keelt. Selle osa funktsionaalsus piirdub räside arvutamise ja kirje muutmisega. Päästiku defineerimiseks tuleb vastavas andmebaasis käivitada SQL laused, mis on toodud lisas (*vt Lisa X).*

Räsi saatmiseks välisele osapoolele kasutatatakse siiski C keeles loodud funktsionaalsusest. See päästik peab igal juhul käivituma peale eelnevalt defineeritud päästikut.

4.2. Andmete digiallkirjastamine

Tegemist on lõputöö ajaliselt kõige mahukama osaga. Mahukaks tegi selle osa see, et digiallkirjastamiseks on mitu tehnilist võimalust ning nende kõigi kasutamisel oli vähemalt lõputöö tegemise ajal omajagu probleeme. Alustades sellest, et Java teek *digidoc4j* algselt ei töötanud vastupidiselt ootustele kasutataval Groovy/Grails platvormil ning sellel puudus kogu vajaliku funktsionaalsuse (nt. PKCS11) tugi. Sellel oli ka muid puudusi: *beta* arendusstaatus, fataalse vea esinemine vähemalt ühe testitud virtuaalse krüptoseadmega (SoftHSM) ning autori arvates liiga keeruline konfigureerimine. Nende probleemide vältimiseks kaaluti kasutada eraldi C/C++ adapterprogrammi, mis suudaks suhelda krüptoseadmega läbi *libdigidocpp* teegi, kuid eesmärgiks võeti siiski *digidoc4j*

kasutamine.

4.2.1. Konteinerid, signatuurid ja signeerimismeetodid

Tehniliselt ei allkirjastata üksikuid dokumente/faile vaid konteinereid. Failide allkirjastamine tähendab nende lisamist konteinerisse ja konteineri allkirjastamist. Eestis kasutatakse kahte tüüpi konteinereid: DDOC ja BDOC, millest viimane jaguneb veel kaheks - *BDOC-TM/ASIC-E LT-TM* ja *BDOC-TS/ASIC-E LT*. (Sertifitseerimiskeskus AS, 2015) Kuna DDOC konteinerite kasutamine pole soovitatav 2015. aastast ning need peaksid kasutuselt kaduma, siis ei tegelenud autor selle formaadi uurimisega. Sertifitseerimiskeskus AS, mis tegeleb digiallkirjastamise infrastruktuuri haldamise ja arendamisega, soovitab siseriiklikke dokumente allkirjastada BDOC-TM (*Time Mark*) allkirjaga, mistõttu ei ole ka BDOC-TS (*Time Stamp*) konteinerid skoobis.

BDOC konteinerid on ZIP-formaadis failid, mis sisaldavad kindla ülesehitusega failide struktuuri. Kõik allkirjastatavad failid asuvad konteineri "juurkaustas". Kõik signatuurid asuvad konteineri META-INF kaustas failinimega *signatureX.xml*, kus *X* on signatuuri indeks (järjekorranumber alates 0-st).

Signatuurid on XAdES (ingl.k. *XML Advanced Electronic Signature*) formaadis. Need sisaldavad andmeid allkirjastaja, tema sertifikaatide, allkirjastamise aja ja allkirja kehtivuskinnitusserveri (OCSP serveri) kohta.

Allkirjastamise protsess on laias laastus selline:

1. Konteineri loomine ja failide lisamine konteinerisse
2. Konteineris olevatest andmetest räsi arvutamine
3. Krüptoseadme sessiooni avamine või salajase võtme faili dekrüpteerimine kasutaja sisestatud PIN koodiga.
4. Räsi saatmine krüptoseadmesse või krüpteerimine failist loetud võtmega
5. Krüptoseadmega allkirjastamisel "allkirja" lugemine seadmest
6. Allkirjale kehtivuskinnituse võtmine (OCSP päring või muu sarnane tehnoloogia)
7. Allkirja vormindamine ja lisamine konteinerile.

4.2.2 *libdigidocpp*

C-keeles kirjtuatud *libdigidoc*'i tugi lõppeb 2015. aasta detsempris. Selle asemele on loodud C++ keeles teek *libdigidocpp*. (Sertifitseerimiskeskus AS, 2015) See on täielik DigiDoc'i implementatsioon, pakkudes toetust kõikidele kasutatavatele DigiDoc'i signatuuride- ja konteineriformaatidele ning erinevatele signeerimismehhanismidele (PKCS11, PKCS12, OpenSSL).

4.2.3. *digidoc4j*

Vana Java teegi *jdigidoc* arendamine lõpetati 2015 aasta juunis ning tugi on samuti lõppemas. (Sertifitseerimiskeskus AS, 2015) Selle asemel on uuem Java teek *digidoc4j*, mis on küll *beta* arendusjärgus, kuid aktiivselt arenduses.

Kuna rakenduste arendamiseks kasutatav Groovy baseerub Java programmeerimiskeelel, on teoreetiliselt võimalik kasutada Java teeke Groovy rakendustes. Siiski ei olnud algselt võimalik teeki muutmata kujul kasutada Groovy/Grails platvormil. Probleemiks olid teatud klassimuutujate läbivalt vigased deklaratsioonid teegis. Teek kasutas mõnede objektide kloonimiseks serialiseerimist, kuid Groovy loodud baitkood ei suutnud oma dünaamilise ehituse tõttu deserialiseerimisel leida serialiseeritud objektide klasse. Isegi, kui see probleem ei oleks osutunud fataalseks, oli tegemist olulise jõudluse piirajaga, sest kõnealuseid muutujaid ei oleks üldse pidanud serialiseerima ning kogu see protsess ei ole ühekordne tegevus. Autor tegi vastava paranduse *digidoc4j* ametlikku koodibaasi.

Veel osutus oluliseks probleemiks ametliku PKSC11 toe puudumine, mis oli eelduseks, et rakendus saaks turvaliselt allkirjastamist läbi viia. Teegil on ainult PKSC12 tugi, mis aga ei ole nii turvaline, sest sel juhul hoitakse allkirjastaja salajast võtit failis, millest on seda võimalik PIN koodi teades lugeda. PKCS11 puhul hoitakse võtit krüptoseadmes, kust kontseptsiooni järgi ei ole seda kuidagi võimalik lugeda. Teegil on siiski mitteametlik ning kohati ebastabiilne PKCS11 tugi. Nimelt kasutab *digidoc4j* alusena modifitseeritud *sd-dss* teeki, mis toetab PKCS11 standardit analoogselt PKCS12-ga. Kuid teadmata põhjusel ei tööta see lahendus, kui arendaja kasutab räsialgoritmi SHA256 (mis on vaikimisi kasutatav algoritm), kusjuures teiste algoritmidega lahendus töötab. Hoolimata vea põhjuse leidmisele kulutatud pikast ajast ei suutnud autor tuvastada vea põhjust ja olemust. Autori arvates on viga parandatud *sd-dss* originaalkoodibaasis, kuid kuna *digidoc4j* ei toeta ametlikult PKCS11 protokolli, ei ole seda veaparandust modifitseeritud

versioonis tehtud.

On ka väiksemaid probleeme *digidoc4j* teegiga. Need puudutavad peamiselt koodi kvaliteeti. Kohati ei peeta kinni üldtunnustatud muutujate, meetodite ja klasside nimekonventsioonist. Näiteks klass *PKCS11SignatureToken* peaks konventsiooni järgi olema nimega *Pkcs11SignatureToken* ning meetod *addTSLSource* nimega *addTslSource*, samas mitte igal poole ei eksita selle reegli suhtes. Lisaks on osa äriloogikat kirjutatud *Configuration* klassi, mis teeb rakendusespetsiifilise konfiguratsioonimehhanismi loomise väga keeruliseks. Grails'il on äriklassi rakenduste jaoks hea konfiguratsioonimehhanism, kuid selle probleemi tõttu ei saa seda kasutada *digidoc4j* teegi konfigureerimiseks.

Ka seda probleemi püüdis autor parandada, kloonides (ingl.k. *fork*) ametlikust repositooriumist isikliku repositooriumi, kus konfiguratsiooniklassist on eraldi välja toodud abstraktne (ingl.k. *abstract*) klass mis sisaldab konfiguratsiooniklassi meetodeid, mille implementatsioon ei ole ülejäänud teegi seisukohalt oluline. Selle "täiustuse" raames tuli välja veel üks viga teegis, mille tingis Java *Reflection* tehnoloogia kasutamine, mille tõttu ei leidnud teek klassi *CustomContainer* implementatsioonidest konstruktorit, mis aksepteeriks parameetrina klassi *Configuration* (autori repositooriumis *AbstractConfiguration*) alamklassi instantsi. Ka selle vea aitas autor parandada, tehes vastava muudatuse ametlikku repositooriumisse.

4.2.3. Digiallkirjastamise kasutamine

Selle peatüki peamiseks väljundiks on tehtud parandused *digidoc4j* teegis. Need parandused võimaldavad kasutada teeki ettevõtte rakendustes ning aitavad ka kaasa teegi ja selle koodi kvaliteedi tõstmisel.

Ettevõtte jaoks on loodud digiallkirjastamise näidisprogrammid ning PKCS11 allkirjastamise prototüüp, mida saab kasutada infosüsteemide arendamisel. Digiallkirjastamise koodi näidis on toodud välja lisades (*vt. Lisa X*).

Autor aitas ka *digidoc4j* teeki integreerida ühte uude ettevõtte arendatavasse infosüsteemi. Oluliseks probleemiks oli see, et *digidoc4j* ei toeta populaarset Java pakihaldussüsteemi, mida infosüsteem kasutas ning

Kokkuvõte

Lõputöö käigus uuriti erinevaid tehnilisi lahendusi, mida saaks kasutada andmete kõrge terviklikkuse tagamiseks. Lahenduste eesmärk oli rakendada ISKE HT.10 ja HT.34 meetmed ning tagada andmebaasis hoitavate andmete jälgitavus. Lõputöö käigus on valminud mitmeid teste jõudluse mõõtmiseks ja vigade ära hoidmiseks.

ISKE meede HT.10 käsitleb andmete räsiaheldamist. Antud lõputöö kontekstis tähendab see kõrge terviklusega andmeid hoidvate andmebaasitabelite kirjete räsiaheldamist. Räsiahela kontrollimise mehhanismi loomine ei ole antud lõputöö skoobis ning autor piirdus ainult selle teoreetilise osa uurimisega. Võimalusi selle meetme rakendamiseks leiti mitmeid ja osad nendest ka realiseeriti.

HT.34 teeb kohustuslikuks kõrge terviklikkuse nõuetega dokumentide digiallkirjastamise vastavalt Eesti Vabariigi digiallkirja seadusele. See esmapilgul triviaalne vajadus osutus tegelikult palju keerulisemaks. Ka siin oli potensiaalseid lahendusi mitu, kuid nendel kõigil olid omad puudused. Näiteks ei olnud java teegil ametlikku PKCS11 tuge ning eraldi teises keeles kirjutatud programmi kirjutamine ja kasutamine oleks olnud palju keerulisem ning võimalik, et ebastabliisem ning seega ebasoovitav. Lõpuks suudeti minna seda teed, et kasutada java teeki mitteametliku PKCS11 toega.

Ka on autor teinud kaks parandust suhteliselt uude DigiDoc'i java teeki nimega *digidoc4j*, mis oli veel lõputöö alguses beta arengujärgus. Esimene muudatus parandas koodis ebaühtsuse, mis üllatavalt põhjustas üsna raskesti tuvastatava probleemi. See probleem ei võimaldanud kasutada teeki ettevõtte poolt kasutatava Grails raamistikuga.

Teine parandus oli veaparandus, mis ei olnud küll ettevõtte poolt arendatava tarkvara jaoks oluline, kuid kuna see viga tuli teegi arenduse käigus välja, siis oli see vaja lihtsuse huvides ära parandada. Siiski ei olnud see veaparandus perfektne, sest autor ei pidanud

vajalikuks kontrollida ühte kasutusjuhtu. See oli esimene kord autori jaoks, kui *test-driven* arendusmeetodit (ing.k. *Test-Driven Development* – TDD) näitas oma reaalset vajalikkust.

Uurimuse tulemust kasutatakse vähemalt ühe uue infosüsteemi loomisel, millel on kõrge tervikluse ja käideldavuse nõue.

Allikad

Autor (aasta). pealkiri [link]

bla bla bla... TODO: viited

ISKE

Riigi Infosüsteemi Amet (2015), “Infosüsteemide turvameetmete süsteem ISKE” [https://www.ria.ee/ee/iske.html]

Riigi Infosüsteemi Amet (2014), “ISKE kataloogid” [https://www.ria.ee/public/ISKE/ISKE_kataloogid/ISKE_kataloogid_7.pdf]

JSON

Phil Haack (2009), “JSON Highjacking”[http://haacked.com/archive/2009/06/25/json-hijacking.aspx]

Ecma International (2013), “The JSON Data Interchange Format” (ECMA-404) [http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf]

Digiallkirjastamine

Sertifitseerimiskeskus AS (2015), “What's the difference between the digital signature formats .ddoc, .bdoc and .asice?” [http://www.id.ee/index.php?id=37370]

Sertifitseerimiskeskus AS (2015), “DigiDoc teegid - üldinfo ja ülevaade” [http://www.id.ee/index.php?id=30290]

PostgreSQL

The PostgreSQL Global Development Group (2015), “PostgreSQL 9.4.5 Documentation” [http://www.postgresql.org/docs/9.4/static/index.html]

Hitoshi Harada (2015), “PL v8” [http://pgxn.org/dist/plv8/doc/plv8.html]

GitHub Pull Requests

Kapp (2015), “Make logger declarations consistent (final static)” [https://github.com/open-eid/digidoc4j/pull/6]

Kapp (2015), “Fix creation of custom container with custom configuration” [https://github.com/open-eid/digidoc4j/pull/8]

Lisad

README.md

```
PostgreSQL trigger, mis räsiaheldab sisestatavaid kirjeid ja/või
saadab tulemuse syslog serverile kujul "tabeli_nimi rea_id hash eelmine_hash"
või kujul "tabeli_nimi rea_id hash" sõltuvalt sellest, kas trigger arvutab räsid
ise või mitte.

Repos olevad *hashchain-trigger-*.so* faild on kompileeritud PostgreSQL 9.4 jaoks
Ubuntu 14.04 x64 masinal.

### Kompileerimine

    make

Väljundiks on kolm faili:
 * hashchian-trigger-default.so – räside arvutamine ja logisse saatmine
 * hashchain-trigger-usecache.so – räside arvutamine ja logisse saatmine; kasutatakse vahemälu
 * hashchain-trigger-logonly.so – ainult räside saatmine logisse

#### Triggeri seadistamine

Stored procedure loomine

    CREATE FUNCTION "hashchainTrigger"() RETURNS trigger
       AS '/path/to/hashchain-trigger-*.so' – failinimi sõltuvalt soovitavast funktsionaalsusest
       LANGUAGE C;

Räsiaheldamise trigger

    -- tabel peab sisaldama välja hashchain, mis on piisava suurusega
    -- et räsi salvestada
    CREATE TRIGGER hashchain BEFORE INSERT ON tabeli_nimi
       FOR EACH ROW EXECUTE PROCEDURE "hashchainTrigger"();

Ainult räside saatmine

    -- tabel peab sisaldama välja hashchain
    CREATE TRIGGER hashchain AFTER INSERT ON tabeli_nimi
       FOR EACH ROW EXECUTE PROCEDURE "hashchainTrigger"();
```

Makefile

```
default:
       gcc trigger.c hashRecord.c \
       I/usr/include/postgresql/9.4/server \
       lpq -lcrypto -shared -fPIC -o hashchain-trigger-default.so

       gcc trigger.c hashRecord.c hashtable.c \
       -DUSE_CACHE -I/usr/include/postgresql/9.4/server \
       -lpq -lcrypto -shared -fPIC -o hashchain-trigger-usecache.so

       gcc trigger.c shipHash.c \
       -DLOG_ONLY -I/usr/include/postgresql/9.4/server \
       -lpq -shared -fPIC -o hashchain-trigger-logonly.so
```