LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ELEKTRONISKO PARAKSTU ĪSTENOŠANA PDF DOKUMENTOS

BAKALAURA DARBS

Autors: **Aleksandrs Goļenkovs**
Studenta apliecības Nr.: ag10097
Darba vadītājs: M. dat. Artūrs Lavrenovs

RĪGA 2015

ANOTĀCIJA

Šis darbs apskata elektronisko parakstu īstenošanu PDF dokumentos. Tas pēta elektronisko parakstu struktūru un funkcionalitāti. Darba mērķis ir izpētīt elektronisko parakstu standartus PDF dokumentiem, struktūru, darbības principus un īstenošanas iespējas. Uzmanība tiek pievērsta pakarstu struktūrai, un ir apskatīti nepieciešamie dati elektronisko parakstu veidošanai.

Tiek izpētīti papildus procesi un servisi, kurus ir nepieciešams nodrošināt elektronisko parakstu sniedzēju pusē. Papildus tiek apskatīti dažu elektronisko parakstu sniedzēju produkti.

Darbā tiek aprakstīta tīmekļa lietotnes veidošana, kas nodarbojas ar parakstu pievienošanu un parakstu pārbaudi PDF dokumentos. Tiek parādīts, kā izstrādātājs var viegli izstrādāt lietotnes, kuras spēj darboties ar elektroniskiem parakstiem PDF dokumentos.

Atslēgvārdi: elektroniskie paraksti, PDF dokumenti, elektronisko parakstu sniedzēji, sertifikāti.

ABSTRACT

Digital signature implementation in PDF documents

This work examines electronic signature implementation in PDF documents. It explores structure and functionality of electronic signatures. The purpose of this work is to research electronic signature standards for PDF documents, structure, principles of operation and implementation possibilities. Focus is on electronic signature structure and information which is required for signature itself.

The paper examines additional processes and services which should be provided from signature provider side. Additionally it researches few signature providers and their products.

This work describes web application creation, which provides functionality to sign and validate PDF documents. It shows how easy software developer can create application which would be able to work with digital signatures inside PDF documents.

Keywords: Digital signatures, PDF documents, digital signature providers, certificates.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

PDF	Portable Document Format – atvērtais elektronisko dokumentu standarts, kas tiek izmantots dokumentu reprezentācijai neatkarīgi no programmatūras, aparatūras vai operētājsistēmas.
PKCS#7	Cryptographic Message Syntax Standard – standarts digitāliem parakstiem, īssavilkumu algoritmiem un datu šifrēšanai,
PKCS#12	Personal Information Exchange Syntax Standard – kriptogrāfiskais standarts, kas definē datņu formātu, kuru izmanto privātās atslēgas un sertifikātu glabāšanai, tas tiek aizsargāts, izmantojot uz paroles bāzētās simetriskās atslēgas.
PKCS#13	Elliptic Curve Cryptography Standard – kriptogrāfiskais standarts, kas publiskām atslēgām izmanto eliptisko līkņu algebriskās struktūras.
CRL	Certificate Revocation List – sertifikātu, jeb sertifikātu seriālo numuru saraksts, kas satur sertifikātu sniedzēja atsauktos sertifikātus, kuriem vairs nevar ticēt.
OCSP	Online Certificate Status Protocol – interneta protokols, kas tiek izmantots sertifikāta atsaukšanas statusu iegūšanai.
TSA	Timestamping Authority – laikspiedolu serviss, kas nodarbojas ar uzticamo laikspiedolu ģenerēšanu.
FIPS	Federal Information Processing Standards – publiski pieejams standarts valsts iestādēm, kuru izstrādāja ASV federālā valdība.

IEVADS

Šobrīd pasaule iet tādā virzienā, ka uzņēmumi un valsts iestādes cenšas pāriet no papīra dokumentiem un elektroniskiem. Jebkura organizācija cenšas automatizēt ikdienas procesus, izmantojot programmatūras risinājumus. Turklāt visbiežāk situācija ir tāda, ka papildus parastai datu glabāšanai ir nepieciešama iespēja dokumentus parakstīt un pārbaudīt.

Šodien eksistē ļoti dažādu elektronisko dokumentu veidi un tipi, katram no šiem dokumentiem ir sava darbības sfēra un savi mērķi. Darboties ar vairākiem dokumentu tipiem var būt diezgan grūti, tāpēc bieži vien lietotāji cenšas piesaistīties kādai konkrētai dokumentu tipu kopai, kas atbilst visām viņu prasībām.

PDF dokumenti ir viens no izplatītākajiem dokumentu tipiem, kas pilnībā atbalsta elektroniskos parakstus. Tas bieži vien tiek izmantots, veidojot dažāda veida atskaitēs, rēķinos, grāmatās un dokumentācijās. To lielākās priekšrocības ir tādas, ka tos ir ļoti viegli izveidot un apskatīt jebkurā ierīcē. Šobrīd gandrīz katra programma, kas nodarbojas ar dokumentu izstrādi, atbalsta PDF dokumenta saglabāšanas funkciju, kas dod iespēju gandrīz jebkuru dokumentu konvertēt PDF dokumentā.

Katru gadu pieprasījums pēc elektroniskiem parakstiem aug un ar to aug arī piedāvāto risinājumu skaits. Neskatoties uz to, ka eksistē jau vairāki elektronisko parakstu risinājumi, un to galvenā ideja ir vienmēr vienāda, parakstīt dokumentu un pārbaudīt parakstu dokumentā, katram risinājumam ir sava specifika un pieeja parakstīšanas procesam. Tātad, izvēloties elektroniskā parakstu sniedzēju ir pilnīgi jāapzinās: kāda parakstīšanas pieeja tiek izmantota, kā sertifikāti tiek glabāti un kā parakstus var pārbaudīt. It īpaši tas ir svarīgs, ja ir nepieciešamība integrēt elektronisko parakstu atbalstu kādā projektā. Turklāt var rasties situācijas, kad programmatūras izstrādātājam ir nepieciešams izveidot jaunu dokumentu parakstīšanas sistēmu. Lai to izdarītu efektīvi un kvalitatīvi, ir nepieciešams zināt visas pieejas, jo šodien nepastāv neviena universāla pieeja šīs problēmas risināšanai.

Šī darba mērķis ir izpētīt elektronisko parakstu būtību un struktūru, pielietojumu un iespējamos parakstīšanas paņēmienus. Tiek identificēti visi nepieciešamie dati elektroniskā paraksta veidošanai. Darba ietvaros tiek veikts vairāku elektronisko parakstu sniedzēju un viņu risinājuma salīdzinājums. Papildus tam tiek aprakstīta tīmekļa lietotnes veidošana, PDF dokumenta parakstīšanai un pārbaudei. Turklāt tiek apskatīti vairāki iespējamie lietotnes uzlabojumi.

Darba struktūra

Darba 1. nodaļā tiek aprakstīti elektronisko parakstu galvenie principi un vairāki kritēriji, kas ir izvirzīti priekš elektroniskiem parakstiem.

Darba 2. nodaļa apraksta elektronisko parakstu īstenošanu PDF dokumentos un tiek apskatīti visi iespējamie parakstu veidi.

Darba 3. nodaļā tiek apskatīti vairāki elektronisko parakstu sniedzēji un tiek izpētīti elektroniskie paraksti, kurus viņi piedāvā.

Darba 4. nodaļā tiek izpētīti rīki, kas ir nepieciešami, lai realizētu elektronisko parakstu atbalstu.

Darba 5. nodaļā tiek apkopoti iegūtie rezultāti un tiek veikti secinājumi.

1. ELEKTRONISKO PARAKSTU IDEJA

Šodien datu aizsardzība ir viena no lielākajām problēmām. Ikdienas dzīvē mēs bieži sastopamies ar situācijām, kad ir nepieciešams būt pārliecinātam par to, ka dokuments vai ziņa ir saņemta tieši no konkrēta cilvēka un dati atbilst oriģinālam. Turklāt šodien gandrīz visi uzņēmumi cenšas pāriet no papīra dokumentiem uz elektroniskiem. Tādēļ rodas vajadzība ieviest parakstus datorikā.

Elektronisko parakstu galvenā ideja ir parādīt, ka persona kas saka, ka ir uzrakstījusi kādu ziņu, realitātē ir šīs ziņas autors. Bieži vien elektroniskos parakstus salīdzina ar parastu rokas parakstu, kas tiek asociēts ar konkrētu personu. Biznesa procesos elektroniskie paraksti tiek izmantoti, lai apstiprinātu datus.

Vairākās pasaules valstīs elektronisko parakstu spēks atbilst parastiem rokas parakstiem. Tomēr katra valsts vai organizācija var izvirzīt savas konkrētās prasības priekš elektroniskiem parakstiem. Latvijā elektronisko dokumentu prasības ir aprakstītas elektronisko dokumentu likumā. Pēc šī likuma drošam elektroniskam paraktam ir jāatbilst sekojošiem kritērijiem:

- Drošs elektroniskais paraksts ir piesaistīts tikai parakstītājam.
- Drošs elektroniskais paraksts nodrošina parakstītāja personas identifikāciju.
- Drošs elektroniskais paraksts ir radīts, izmantojot drošus elektronisko parakstu radīšanas līdzekļus, kurus var kontrolēt tikai parakstītājs.
- Drošs elektroniskais paraksts ir saistīts ar parakstāmo dokumentu tā, lai vēlākās izmaiņas šajā dokumentā būtu pamanāmas.
- Drošs elektroniskais paraksts ir apliecināts ar kvalificētu sertifikātu.

Latvijā oficiāli tiek atzīts viens elektronisko parakstu sniedzējs Latvijas Valsts radio un televīzijas centrs [1].

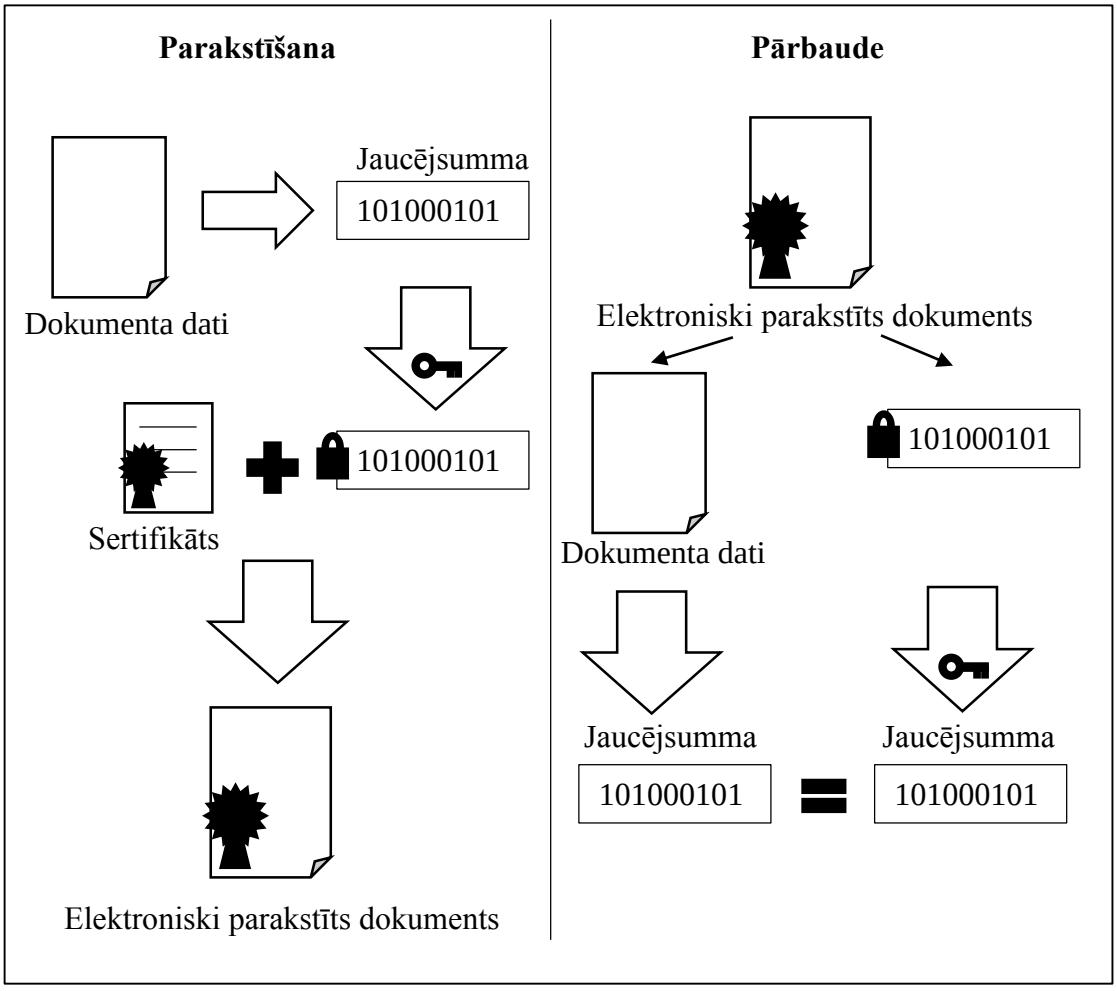
Mūsdienās elektroniskais paraksts ir ļoti plašs termins un tas iekļauj sevī vairākus dažādu elektronisko parakstu veidus, kas atšķiras pēc savas funkcionalitātes un drošības kritērijiem. Katrs elektroniskā paraksta veids balstās uz atsevišķas tehnoloģijas un uz šo dienu eksistē trīs galvenās parakstu tehnoloģijas.

1.1. Kriptogrāfiskie paraksti

Kriptogrāfiskie paraksti jeb digitālie paraksti izmanto kriptogrāfiskās metodes. Šīs metodes nodrošina oriģinālās ziņas viengabalainību un autentiskumu. Kriptogrāfiskie paraksti bieži vien tiek izmantoti kā standarts parakstot dokumentus.

Visbiežāk kriptogrāfiskie paraksti izmanto privātās un publiskās atslēgas dokumenta parakstīšanai un pārbaudei. Parakstot dokumentu, tiek izveidots visu datu jaucējsumma. Šis

kods tiek iešifrēts, izmantojot parakstītāja privāto atslēgu. Tālāk sertifikāts ar jaucējsummu tiek pievienots dokumentam. Lai pārbaudītu dokumentu, ir nepieciešams izņemt datus un parakstīto jaucējsummu no dokumenta. No datiem atkal tiek izveidota jaucējsumma un parakstītā jaucējsumma tiek atšifrēta, izmantojot publisko atslēgu. Ja abas iegūtās jaucējsummas ir vienādas, tad dati it patiesi.



1.1. att. **Dokumenta parakstīšanas un pārbaudes process**

Šī pieeja balstās uz to, ka publiskām atslēgām jābūt pieejamām dokumenta saņēmējam, lai tam būtu iespēja dokumentu pārbaudīt.

1.2. Biometriskie paraksti

Biometriskie paraksti izmanto cilvēka bioloģiskās identifikācijas īpašības. Šī pieeja balstās uz to, ka datiem tiek pievienoti kādi biometriskie mērījumi. Par biometrisku mērījumu var būt pirkstu nospiedumu skaits, rokas ģeometrija vai acs tīklenes paraugs. [2] Šie mērījumi bieži vien tiek savākti, izmantojot speciālās ierīces. Neskatoties uz to, ka šie mērījumi katram cilvēkam ir unikāli, tos visus var ļoti viegli viltot. Turklāt tās nav iespējams mainīt, ja mērījums tiek kompromitēts. Tāpēc šie paraksti mūsdienās tiek lietoti ļoti reti.

1.3. Digitāli notvertie paraksti

Digitāli notveramie paraksti, jeb dinamiskie paraksti izmanto papildus ierīces, lai notvertu personas rokas parakstu. Šodien šis paraktu veids kļūst diezgan populārs, jo šīs ierīces kļūst pieejamākas un gandrīz jebkurš moderns mobilais telefons atbalsta šādus parakstus.

Ierīces tiek izmantotas, lai konvertētu rokas parakstu, paraksta datos. Šie paraksta dati vēlāk tiek izmantoti, lai identificētu dokumenta izmaiņas, izmantojot dokumenta jaucējsummu. Lai būtu iespēja pārbaudīt parakstu, bieži vien kopā ar pašu rokas parakstu tiek saglabāti papildus dati par pašu parakstīšanas procesu. Neskatoties uz to, izmantojot šādu paņēmienu, paraksta pārbaude kļūst relatīvi grūtāka, jo bieži vien būs nepieciešamība piesaistīt ekspertus, jo tāpat kā parastu rokas parakstu, digitāli notvertos parakstus var viegli viltot.

2. ELEKTRONISKIE PARAKSTI PDF DOKUMENTOS

Adobe PDF dokumenti ir viens no visizplatītākajiem dokumentu formātiem, jo tas piedāvā vairākas priekšrocības:

- Pieejams – apskatīt PDF dokumentu var jebkurš, nepieciešamā programmatūra ir bezmaksas un viegli pieejama.
- Portatīvs – tas ir viegls un var strādāt uz jebkuras platformas: Windows, Mac OS, Linux, Android u.t.t.
- Drošs – ir ļoti viegli uzstādīt paroli uz dokumenta, kas neļaus citiem lietotājiem atvērt tos vai kopēt datus no tiem.
- Digitālie paraksti – digitālie paraksti un sertifikāti ir viens no galveniem kritērijiem biznesa sfērā, tie dod iespēju pārliecināties par datu patiesumu.

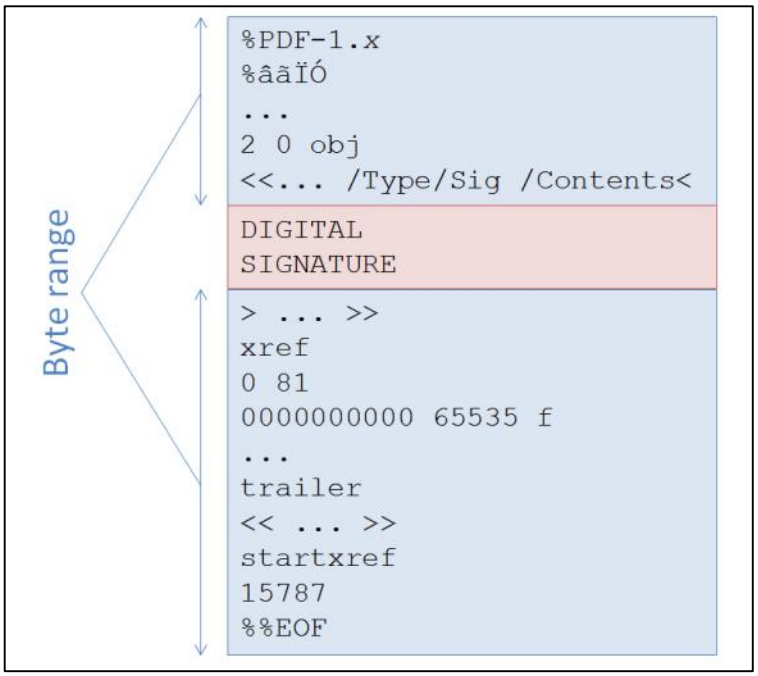
Šobrīd statistika rāda, ka vairāk nekā 600 miljoni PDF dokumentu ir publiski pieejami, kas padara šo formātu vienu no visizplatītākajiem dokumentu formātiem.

Viena no PDF dokumentu priekšrocībām ir iespēja parakstīt dokumentu, izmantojot elektronisko parakstu. Šis paraksts nodrošina to, ka dokuments nebija mainīts pēc paraksta pievienošanas un to, ka mēs varam būt pārliecināti par dokumenta autora identitāti.

Elektroniskais paraksts tiek izveidots, izmantojot speciālus algoritmus un šifrēšanas metodes, kas ir aprakstītas ISO-32000-1 standartā. [3] Šajā darbā mēs papildus apskatīsim specifikācijas, kas ir aprakstītas ISO-32000-2 standartā, kurš tiks publicēts 2016. gadā. [4]

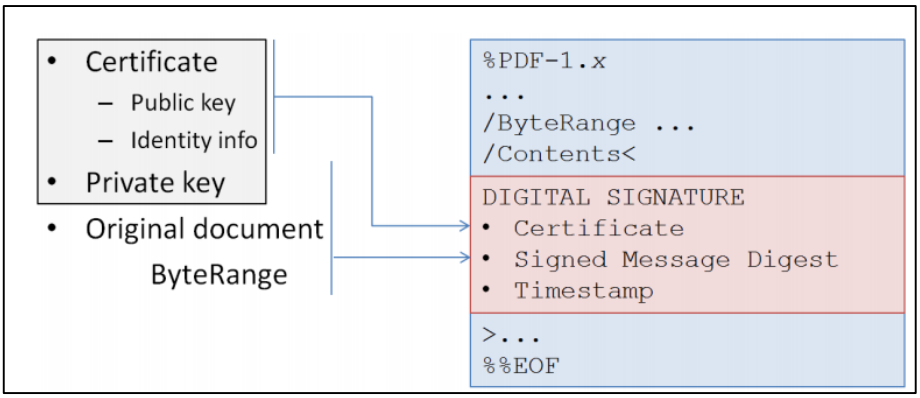
2.1. Elektronisko parakstu struktūra PDF dokumentos

Kad dokuments tiek parakstīts, paraksts tiek pievienots visam dokumentam. Paraksts pārklāj visu dokumentu, izņemot fiksētu bitu masīvu, kurš tiek atstāts pašam parakstam. Ir vairāki veidi, kā var pievienot dokumentam, tomēr neatkarīgi no paņēmiena tiks parakstīts viss dokuments, šodien nav iespējams parakstīt kādas konkrētas dokumenta daļas vai lapas. Shematiski elektroniskais paraksts PDF dokumentā izskatīsies sekojoši (2.1. att.).



2.1. att. **Parakstītā PDF dokumenta shēma**

Lai izveidotu jaunu parakstu, mums ir nepieciešami dati par pašu dokumentu un personu, kas paraksta to. Pirmkārt, mums ir nepieciešams sertifikāts, ar kura palīdzību mēs varēsim identificēt lietotāju. Sertifikātam jāsatur personas informācija un viņa publiskā atslēga. Otrkārt, ir nepieciešams no visa dokumenta datiem izveidot jaucējsummu, un parakstīt to ar personas privāto atslēgu. Tas mums dos iespēju pārbaudīt dokumenta integritāti. Un beigās parakstam tiek pievienots laikspiedols (2.2. att.).



2.2. att. **Parakstam nepieciešamie dati un to pielietojums**

Tomēr tie nav visi dati, kas tiek glabāti parakstā. Tā kā ir nepieciešams ne tikai dokumentu parakstīt, bet arī pārbaudīt to pēc tam, ir nepieciešams saglabāt vairākus papilddatus, kas ir nepieciešami priekš dokumenta validēšanas. Ja atvērt PDF dokumentu, izmantojot PDF satura pārlūku, var redzēt visu informāciju, kas tiek saglabāta kopā ar parakstu (2.3. att.). [5]

```
◦ V (dict) [id: 71, gen: 0]
    ◦ ByteRange (array)
        0 = 0
        1 = 5441
        2 = 9365
        3 = 173396
    Contents = "0\u0004$\u0006\t*H÷
    \r\u0001\u0007\u0002 \u0004\u00150\u0004\u0011\u0002\u0001\u00011\u000f0\r\u0006\t`H\u00
    \r\u0001\u0007\u0001 \u0002m0\u0002i0\u0001Ò \u0003\u0002\u0001\u0002\u0002\n5ÝÏ9@f¹
    +0\r\u0006\t*H÷
    \r\u0001\u0001\u0005\u0005\u00000e1\u001b0\u0019\u0006\u0003U\u0004\u0003\u0013\u0012
    Golenkov1\t0\u0007\u0006\u0003U\u0004\n\u0013\u00001\t0\u0007\u0006\u0003U\u0004\u000
    \r\u0001\t\u0001\u0016\u0014eldar22402@gmail.com1\u000b0\t\u0006\u0003U\u0004\u0006\u0
    Golenkov1\t0\u0007\u0006\u0003U\u0004\n\u0013\u00001\t0\u0007\u0006\u0003U\u0004\u000
    \r\u0001\t\u0001\u0016\u0014eldar22402@gmail.com1\u000b0\t\u0006\u0003U\u0004\u0006\u0
    \r\u0001\u0001\u0001\u0005\u0000\u0003\u00000\u0002\u0000¤EiS÷ú\\°-[|
    Ó2\u001f7ìßk1\u0002Q²ÚÈÝãÐ[ÎK¼Èm5i\"}óÊõ¿÷?þU\fw\f\u0000í&Íiß¶\u0007Ï²-
    <çºmwà|\u000eê&òw#@ú,¡\u0010\u0016=\rï2|õé¢\u001a¤U\\{68¼|«VT«-gµ¨Y^Yi£Ï×Ì´\fÃ
    \u0002\u0003\u0001\u0000\u0001£
    0\u001e0\u000f\u0006\t*H÷/\u0001\u0001\n\u0004\u0002\u0005\u00000\u000b\u0006\u0003U\u
    \r\u0001\u0001\u0005\u0005\u0000\u0003\u0000RÖ©Æê`cX¿°
    ¤¾/CêíX*(Òþ4Zi\u0019=Ù²PI\tõV\u0015¼X\u0004ôÅ\u000bX\u001aø·mè\\Q\u0019kJ¦\u0013½
    \u0017±\u001f3å\u0019{¡%3ç7z|D¸×Ö\u000eÀHI8[«\u0015%¤7c¥\u001eP2ßå
    \no\u0006V5ªÂêa\u0018ÜSyì\"2)5R§¬&ö
    \\1\u0001{0\u0001w\u0002\u0001\u00010s0e1\u001b0\u0019\u0006\u0003U\u0004\u0003\u0013
    Golenkov1\t0\u0007\u0006\u0003U\u0004\n\u0013\u00001\t0\u0007\u0006\u0003U\u0004\u000
    \r\u0001\t\u0001\u0016\u0014eldar22402@gmail.com1\u000b0\t\u0006\u0003U\u0004\u0006\u0
    +0\r\u0006\t`H\u0001e\u0003\u0004\u0002\u0001\u0005\u0000 \\0\u000f\u0006\t*H÷/\u0001\u0
    \r\u0001\t\u00031\u000b\u0006\t*H÷\r\u0001\u0007\u00010/\u0006\t*H÷
    \r\u0001\t\u00041\"\u0004 FEWOÓ<X\f+·\u0002Zä®\u001d=tõ¹cõñ¶:fiÒ0\r\u0006\t*H÷
    \r\u0001\u0001\u000b\u0005\u0000\u0004M\u001a\"£(XéFkÙ;õÀ3<å.Ä·µÍÅ¡Fõñ\u001eTÿÕ¬ú
    \u0011\u0012eok(A\u0019±|0\u00001d}eÔãKL¯ºË\u0017áÅ/ö5ûG\u0014íætõ\tÓ\u000f)?
    \nóW\u0002Åjâ6öÜ+GuG§J³ã×
    $Ð¸ìùDó'%I$\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u
    Filter = /Adobe.PPKLite
    M = "D:20150516193918+03'00'"
    Name = "Alexander Golenkov"
  • Prop_Build (dict)
    SubFilter = /adbe.pkcs7.detached
    Type = /Sig
```


2.3. att. **Elektroniskā paraksta saturs**

Katrs elektronisks paraksts satur 4 obligātos parametrus:

- ByteRange – masīvs no 4 skaitļiem, kas definē parakstītos PDF dokumenta apgabalus [0, bits pirms paraksta, bits pēc paraksta, dokumenta beigas];
- Contents – iekodēti paraksta dati;
- Filter – definē parakstu apdarinātāju, bieži vien tiek izmantots "/Adobe.PPKLite" apdarinātājs;
- SubFilter – definē metodi, kas tiks izmantota paraksta datu šifrēšanai;

Kā arī var būt pievienoti vairāki papildus parametri, atkarībā no izvēlētās programmatūras paraksta pievienošanai. Piemēram: Adobe Reader pievieno papildus parametru "Prop_Build", kur tiek aprakstīta programmatūras versija, kas tiek izmantota paraksta pievienošanai. [6]

PDF standarts atbalsta sekojošos papildus parametrus:

- Type – paraksta objekta tips. Parakstu var pievienot kā dažādus objektus. Ja pievienot parakstu visam dokumenta, paraksta objekta tips būs "/Annot" (Annotācija), bet, ja ievietot parakstu iepriekš definētā paraksta laukumā, tad tās tips būs "/Sig";
- Changes – masīvs, kas satur 3 skaitļus. Pēc pēdējā paraksta izmainīto lapu, lauku un ievadīto datu skaits;

- Name – parakstītāja vārds;
- M – laikspiedols. ISO-32000-1 standarts saka, ka šo parametru nepieciešams aizpildīt tikai tad, kad nav iespējams laikspiedolu pievienot iekodētiem paraksta datiem;
- Location – dators vai vieta, kur tika pievienots paraksts;
- Reason – iemesls paraksta pievienošanai;
- ContactInfo – parakstītāja informācija, lai būtu iespējams ar viņu sazināties un pārbaudīt parakstu.

2.2. Elektronisko parakstu iekodēšana

Kā jau bija parādīts iepriekš, katrs elektroniskais paraksts satur sevī "Contents" parametru, kas satur sevī iekodētus paraksta datus. Šodien, pēc noklusējuma Adobe izmanto PKCS#7 parakstus, kas nosaka, ka "Contents" parametram jāsatur sevī personas sertifikātu, dokumenta jaucējsummu, kas ir parakstīts ar sertifikāta privāto atslēgu, un papildus tam var saturēt sevī laikspiedolu.

PDF dokumenti atbalsta vairākus īssavilkuma algoritmus, kas dokumenta datus konvertē jaucējsumma. Algoritms tiek izvēlēts, balstoties uz elektronisko parakstu standartu, kuru mēs gribam atbalstīt. PDF standarts atbalsta sekojošos īssavilkuma algoritmus:

- SHA1 – kriptogrāfisks īssavilkuma algoritms, kuru izstrādāja ASV Nacionālās drošības aģentūra. 160 bitu jaucējsumma ir uzskatīta par drošāku nekā MD5 algoritms. Tomēr 2005. gadā tika atrastas vairākas drošības problēmas. [7]
- SHA2 – uzlabotais SHA1 kriptogrāfiskais īssavilkuma algoritms, kas izlaboja visas SHA1 problēmas. Šis algoritms atbalsta vairākas jaucējfunkcijas: SHA-224, SHA-256, SHA-384 un SHA-512. Funkciju atšķirība ir izvedamās jaucējsummas garums.
- RIPEMD – kriptogrāfisks īssavilkuma algoritms, kurš tika izstrādāts Leuven Katoliskā Universitātē. Eksistē vairākas algoritma versijas, kur galvenā atšķirība ir jaucējsummas garums: RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320.

Kad no dokumenta datiem ir izveidota jaucējsumma, to ir nepieciešams parakstīt. PDF standarts atbalsta sekojošos parakstīšanas algoritmus:

- RSA – šis algoritms ir nosaukts pēc viņa projektētājiem (Rons Rivest, Adi Shamir, Leonard Adleman). Tas ir plaši izmantots, lai nodrošinātu datu aizsardzību. Iekodēšanas atslēgā ir publiski pieejami un atšķiras no atkodēšanas atslēgas. Publiskā atslēga tiek izveidota, izmantojot divus lielus pirmskaitļus, un paši pirmskaitļi tiek paslēpti. Izmantojot šādu algoritmu tikai persona, kas zina izejas pirmskaitļus, ir

spējīga efektīvi atkodēt datus. [8] Šis algoritms tika standartizēts PKCS#1 standartā 1991. gadā.

- Digitālo Parakstu Algoritms (DSA) – FIPS standarts digitāliem parakstiem. Salīdzinājumā ar RSA, tas ir ātrāks dokumenta parakstīšanai, tomēr lēnāks dokumenta pārbaudei.
- Eliptisko Līkņu Digitālo Parakstu Algoritms (ECDSA) – šis algoritms tika ieviests jaunajā PKCS#13 standartā, kas atrakstīts PDF 2.0. Tas izmanto eliptisko līkņu kriptogrāfiju, kas dod iespēju sasniegt augstāku drošības līmeni, izmantojot īsāku atslēgu. [9]

Parakstot jebkuru PDF dokumentu, būs piedāvāti vairāki dažādi īssavilkuma un parakstīšanas algoritmi. Konkrētas algoritmu versijas ir atkarīgas no šifrēšanas metodes, kas ir definēta paraksta "SubFilter" parametrā. Tabulā 2.1. ir redzami visi apakšfiltri un atbalstītie īssavilkuma un parakstīšanas algoritmi.

2.1. Tabula. **Apakšfiltru salīdzinājums pēc atbalstītajiem algoritmiem**

	Adbe.pkcs7.sha1	**Adbe.x509.rsa_sha1**	**Adbe.pkcs7.deatached, ETSI.CAdES.deatached, ETSI.RFC3161**
Atbalstītie datu īssavilkuma algoritmi	SHA1	No PDF 1.3 – SHA1; No PDF 1.6 – SHA256; No PDF 1.7 – SHA384, SHA512, RIPEMD160	No PDF 1.3 – SHA1; No PDF 1.6 – SHA256; No PDF 1.7 – SHA384, SHA512, RIPEMD160
RSA algoritmi	No PDF 1.3 – līdz 1024 bitiem; No PDF 1.5 – līdz 2048 bitiem; No PDF 1.7 – līdz 4096 bitiem;	No PDF 1.3 – līdz 1024 bitiem; No PDF 1.5 – līdz 2048 bitiem; No PDF 1.7 – līdz 4096 bitiem;	No PDF 1.3 – līdz 1024 bitiem; No PDF 1.5 – līdz 2048 bitiem; No PDF 1.7 – līdz 4096 bitiem;
DSA algoritmi	No PDF 1.6 – līdz 4096 bitiem	Nav atbalstīts	No PDF 1.6 – līdz 4096 bitiem
ECDSA	Nav atbalstīts	Nav atbalstīts	Ir atbalstīts
Komentāri	Pēc ISO-32000-2 šīs apakšfiltrs vairs nebūs atbalstīts.		

Neskatoties uz to, ka PDF atbalsta SHA1 algoritmu, tas vairs nav rekomendēts drošu parakstu veidošanai, un ir rekomendēts izmantot drošākus īssavilkuma algoritmus. [10] Turklāt NIST (US National Institute of Standards and Technology) rekomendē vairs neizmantot RSA 1024 bitu garumā un, izmantots RSA 2048 bitu garumā vai vairāk. Šī rekomendācija tiek atbalstīta vairākās Eiropas valstīs. [11]

2.3. Laikspiedols

Laikspiedols ir simbolu virkne vai iekodētā informācija, kas identificē, kad dokuments bija parakstīts. Tomēr pēc noklusējuma laikspiedols tiek izveidots, balsoties uz lokālajiem laika uzstādījumiem. Ja mums vajadzētu pievienot kādu parakstu pagātnē, mēs varam viegli šādu parakstu viltot, mainot laika uzstādījumus uz datora un parakstot PDF dokumentu. Turklāt izmantojot šādu paņēmienu, mēs varam parakstīt dokumentus ar sertifikātiem, kuriem ir beidzies derīguma termiņš.

Vienīgā iespēja kā mēs varam atrisināt šo problēmu, ir pieprasīt, lai ar katru parakstu laikspiedols ņemtu no Laikspiedola uzlikšanas iestādes (TSA). TSA nodrošina tiešsaistes servisu, kas pievieno laikspiedolu parakstam. Tomēr tas nozīmē, ka katru reizi, kad mēs gribam parakstīt dokumentu, mums ir nepieciešams interneta pieslēgums. Turklāt sazināšanās ar TSA var aizņemt laiku, kas var kļūt par problēmu, ja ir nepieciešams parakstīt daudz dokumentu.

2.4. Digitālo parakstu tipi

ISO-32000-1 nosaka, ka PDF dokumentiem ir nepieciešams atbalstīt 3 standartus paraksta tipus.

Viens vai vairākiem apstiprinājuma parakstiem – tie it tie paraksti, kurus mēs varam pievienot PDF dokumenta paraksta laukiem. To galvenais nolūks ir apstiprināt dokumentu vai dokumenta datus.

Līdz vienas sertifikācijas parakstam – šis paraksts dod iespēju pievienot dokumentam dažādas drošības atļaujas. PDF atbalsta sekojošus sertifikācijas līmeņus:

- Izmaiņas nav atļautas – pēc šāda paraksta pievienošanas nav iespējams to nekādā veidā mainīt, jebkuras izmaiņas salauzīs parakstu.
- Veidlapas aizpildīšana – pēc šāda paraksta pievienošanas būs iespējams aizpildīt dokumenta laukus kā arī pievienot vienu vai vairākus apstiprinājuma parakstus, jebkuras citas izmaiņas salauzīs parakstu.
- Veidlapas aizpildīšana un anotāciju pievienošana - pēc šāda paraksta pievienošanas būs iespējams aizpildīt dokumenta laukus, pievienot dokumenta anotācijas, kā arī pievienot vienu vai vairākus apstiprinājuma parakstus, jebkuras citas izmaiņas salauzīs parakstu.

Līdz diviem lietošanas tiesību parakstiem – šie paraksti tiek izveidoti, izmantojot Adobe instalācijas privāto atslēgu, un šis paraksts tiek izmantots, lai ieslēgtu papildus funkcionalitāti PDF dokumentiem. Kā piemēru var izskatīt bezmaksas Adobe Reader versijas līdz versijai X. Izmantojot šādu Adobe Reader, lietotājiem nebija iespējas parakstīt PDF dokumentus, ja tiem

nebija pievienots lietošanas tiesības paraksts. Lai pievienotu lietošanas tiesību parakstu bija nepieciešams nopirkt kaut vienu Adobe Acrobat produktu un izmantot tā "Reader Enabled" funkcionalitāti, lai pievienotu nepieciešamās tiesības.

2.5. Saistītie pētījumi

Elektronisko parakstu koncepts pirmo reizi tika aprakstīts 1976. gadā pētījumā "Jauns virziens kriptogrāfijā". [12] Kopš tiem laikiem kriptogrāfiskie algoritmi un drošības standarti mainījās vairākas reizes.

Runājot par īssavilkuma funkcijām, jau 2005. gadā. tika publicēts pētījums, kas aprakstīja vairākus iespējamos uzbrukumus SHA1 īssavilkuma funkcijai. Viens no pēdējiem pētījumiem "Ļaunprātīgās īssavilkuma funkcijas" par SHA1 problēmām tika publicēts 2013. gadā. [13] Šis pētījums parāda, kādā veidā var pilnīgi kompromitēt šī algoritma drošību. Lai uzlabotu SHA1 tika izstrādāts SHA2 algoritms, kas galvenokārt darbojās ar garākām atslēgām. 2007. gadā tika publicēts vēl viens pētījums "Uz uzlabotā ceļa priekš SHA-2", kas apskatīja SHA2 veiktos uzlabojumus un pārbaudīja vairākas uzbrukuma metodes.[14] Turklāt tajā pašā laikā tika publicēti vairāki pētījumi, kuru mērķis bija elektronisko parakstu uzlabošana. Viens no šiem pētījumiem bija "Digitālo parakstu uzlabošana ar nejaušo īssavilkumu palīdzību", tas ir veltīts nejaušiem īssavilkumiem un tika publicēts 2007. gadā. [15]

Vairāki pētījumi tika veikti arī saistībā ar RSA parakstu shēmu. Viens no pēdējiem pētījumiem ar nosaukumu "RSA bāzēta nenoliedzamu parakstu metodes pētījums" tika publicēts 2013. gadā, to izstrādāja Xin Li un Chunxilao Liu. [16] Šī pētījuma galvenais mērķis bija analizēt RSA parakstu struktūru un nenoliedzamo parakstu struktūru. Turklāt vairāki pētījumi tika veltīti relatīvi jauniem parakstīšanas algoritmiem. Viens no tiem ir "Jauni vispārējie digitālo parakstu algoritmi", un tas tika publicēts 2011. gadā. Šis pētījums apskata DSA un ECDSA parakstīšanas algoritmus un to priekšrocības. [17]

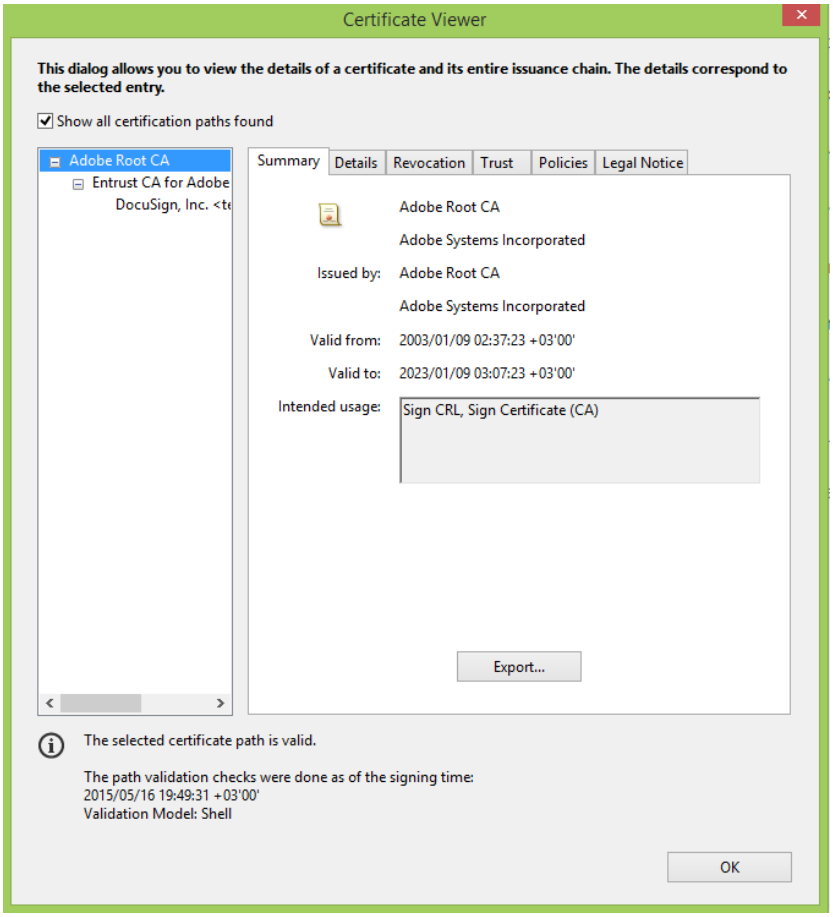
Runājot par pašiem parakstiem un to pielietojumu, 2012. gadā tika publicēts pētījums ar nosaukumu "Digitālo parakstu un to pielietojumu apskats", kura mērķis bija izpētīt elektronisko parakstu pielietojumus un to drošību. Šī pētījuma secinājums bija tāds, ka elektroniskie paraksti turpinās attīstīties un nākotnē var gaidīt vēl drošākas un spēcīgākas elektronisko parakstu sistēmas. [18] Turklāt 2013. gadā, vienas slimnīcas ietvaros, tika veikts pētījums. Pētījuma mērķis bija izpētīt elektronisko parakstu efektivitāti un salīdzināto to ar pašreizējo situāciju. Pētījuma piedalījās 50 dalībnieki: 10 slimnīcas darbinieki un 40 slimnīcas pacienti. Tika izvēlētas 4 dažādas sistēmas, kuru izmantošanai dalībniekiem bija jāparaksta vairākus testa dokumentus. Uzdevuma izpildīšanai nepieciešamais laiks un visas pieļautās kļūdas tika strikti dokumentētas, kā arī katram dalībniekam tika uzdoti vairāki jautājumi par sistēmu. Pētījuma

rezultāti parādīja, ka elektroniskā paraksta izmantošana aizņēma nedaudz mazāk laika, nekā tradicionālā parakstīšana. Neskatoties uz to, par jauno sistēmu, dalībnieki atsaucās ļoti pozitīvi, jo tā deva viņiem drošības sajūtu un iespēju vieglāk apskatīt parakstāmos dokumentus. [19]

3. ELEKTRONISKO PARAKSTU SNIEDZĒJI UN TO LOMA

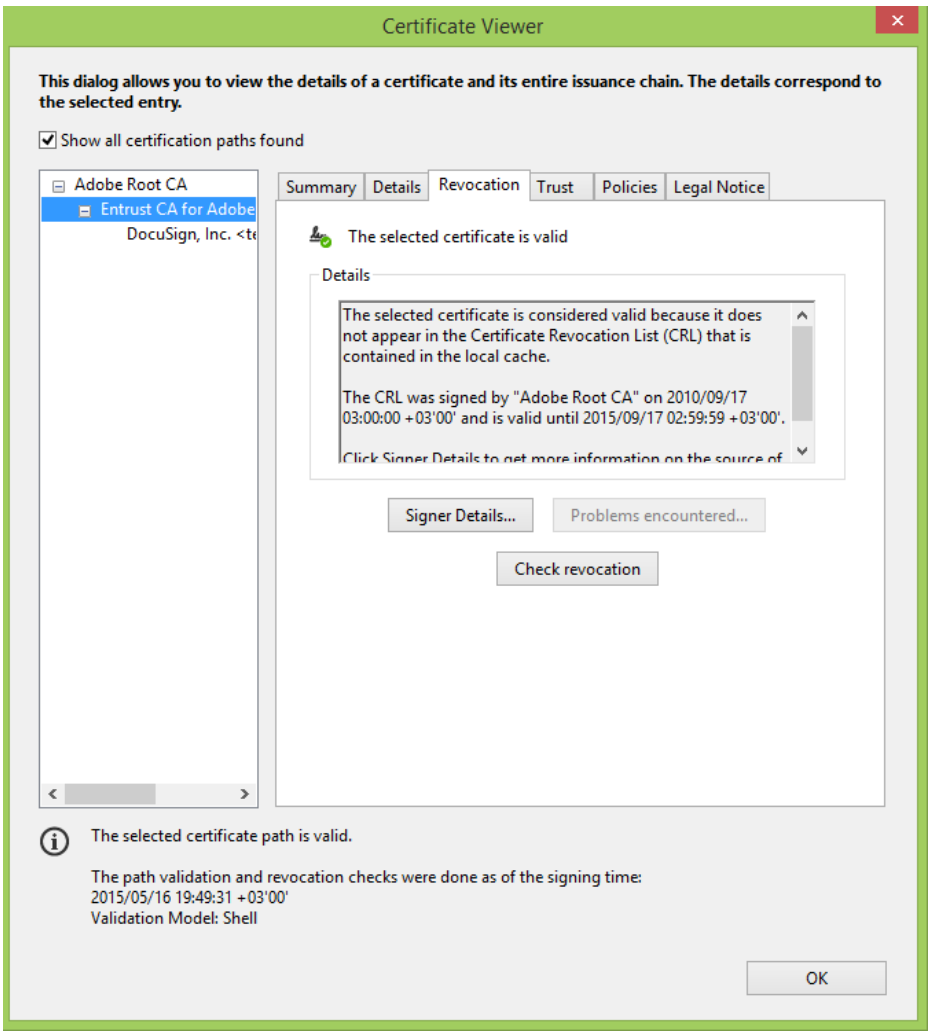
Šodien bieži vien rodas situācijas, kad mums ir nepieciešams aizsūtīt parakstīto dokumentu kādai citai personai, kura mūs var nepazīt. Neskatoties uz to, ka kopā ar mūsu parakstu būs aizsūtīts arī publiskais sertifikāts, šim sertifikātam nevar pilnībā ticēt, jo parakstīt dokumentu var jebkurš un sertifikātu var izveidot arī jebkurš. Lai atrisinātu šo problēmu, ir nepieciešams starpnieks, kuram ticētu abi procesa dalībnieki. Starpnieka vietā bieži vien atrodas kāda organizācija, kura nodarbojas ar sertifikātu pārdošanu. Tās galvenais mērķis ir nodrošināt, ka persona, kas parakstīja dokumentu ir tā persona, kurai vajadzēja to parakstīt.

Elektronisko parakstu sniedzēji bieži vien piedāvā kādu konkrētu sertifikātu, kas tiks asociēts ar konkrētu personu. Tomēr, parakstot dokumentu ar šādu sertifikātu, var redzēt, ka dokuments satur sevī sertifikātu ķēdi, nevis tikai parakstītāja sertifikātu. Šī ķēde tiek izveidota, parakstot katru sertifikātu ar tā vecāko sertifikātu (3.1. att.).



3.1. att. **Sertifikātu ķēde PDF dokumentam**

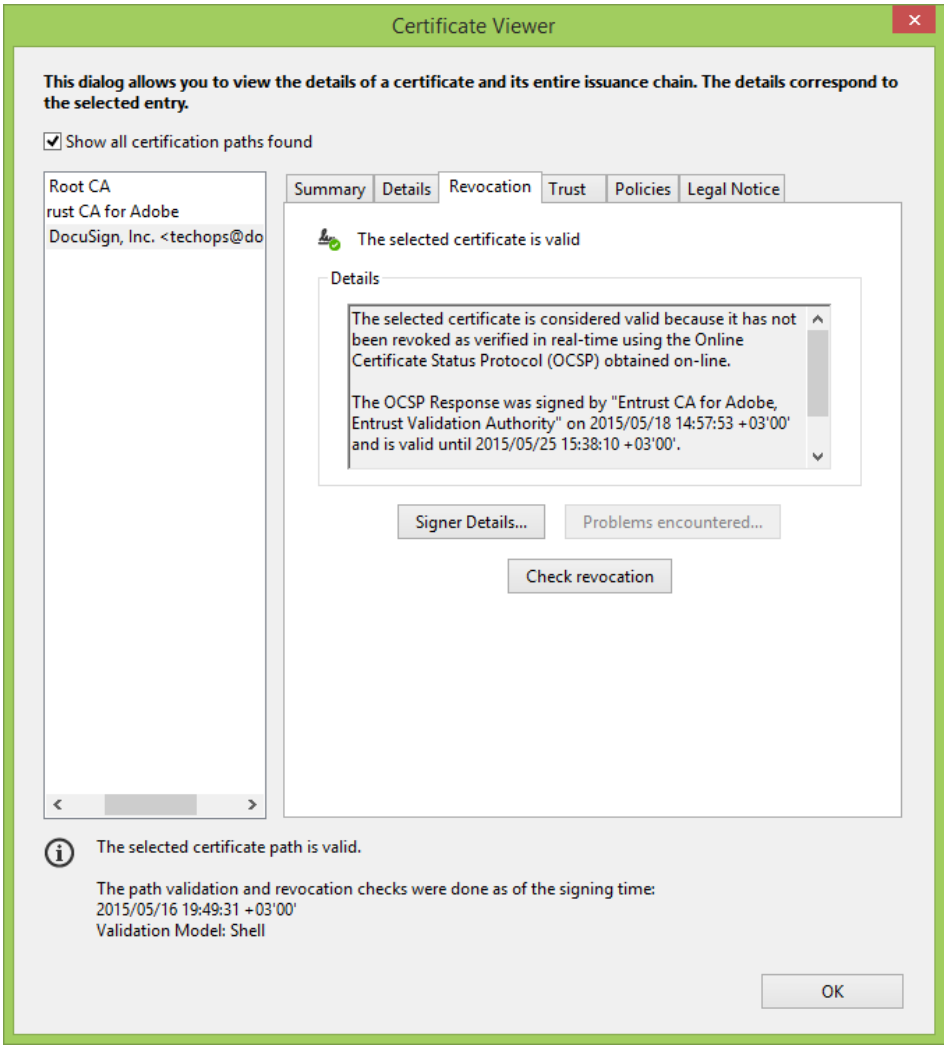
Tomēr, neskatoties uz visiem sertifikātiem, pastāv viena liela problēma. Neskatoties uz visiem drošības līmeņiem, var izveidoties situācija, kad sertifikāts kļūst kompromitēts. Šajā situācijā mums kaut kur paliek sertifikāts, ar kuru vēl iespējams parakstīt dokumentus.



3.2. att. **Sertifikātu pārbaude, izmantojot CRL**

Lai atrisinātu šo problēmu, tiek izveidots tā sauktais Sertifikātu Atsaukšanas saraksts (CRL), kurš satur visus sertifikātus, kas tika atsaukti kādu iemeslu dēļ. Eksistē divi varianti, kā šo sarakstu var pievienot PDF dokumentam: visu sarakstu var ievietot PDF dokumentā vai pievienot dokumentam saiti uz sertifikātu atsaukšanas sarakstu. Izmantojot šo saiti, ir iespējams automātiski pārbaudīt vai sertifikāts ir derīgs (3.2. att.).

CRL nav vienīgais risinājums šādai problēmai. Alternatīva būtu izmantot Tiešsaistes Sertifikātu Statusu Protokolu (OCSP). Ar šī protokola palīdzību ir iespējams pārbaudīt sertifikāta statusu tiešsaistē, izdarot pieprasījumu serverim. Izmantojot šo protokolu, lietotājs aizsūta pieprasījumu ar sertifikāta datiem OCSP serverim un saņem vienu no četrām atbildēm: “OK”, “Atsaukts”, “Nezināms” vai “Kļūda”. Gandrīz visi modernie interneta pārlūku atbalsta OCSP protokolu, atskaitot “Google Chrome”. [20]



3.3. att. **Sertifikātu pārbaude, izmantojot OCSP**

Izmantojot šo pieeju ir ērtāk, jo vairs nav vajadzības glabāt dokumentā nekādu sarakstu, kas var aizņemt diezgan daudz vietas. OCSP atbilde ir konstanta garuma, un to ir viegli piestiprināt PDF dokumentam.

3.1. Elektronisko parakstu sniedzēju salīdzinājums

Mūsdienās eksistē vairāki risinājumi un elektronisko parakstu sniedzēji. Šajā darbā tiks apskatīti tikai daži elektronisko parakstu sniedzēji un tie ir izvēlēti, baltoties uz dažādu risinājumu arhitektūru.

"EchoSign" ir elektronisko parakstu risinājums, kuru izstrādāja "Adobe EchoSign" 2005. gadā. Pēc savas būtības tā ir tīmekļa lietotne, kas paraksta dokumentus servera pusē. Jebkurš dokuments tiek augšupielādēts uz "EchoSign" serveriem, kur dokumentam tiek pievienots elektroniskais paraksts. Dokumenti tiek parakstīti, izmantojot tikai "EchoSign" sertifikātu ķēdi. Turklāt ķēde nesatur individuālo lietotāja sertifikātu, jo tas netiek ģenerēts. Pats paraksta objekts nesatur nekādu informāciju par dokumenta parakstītāju, izņemot parakstītāja ievadīto paraksta attēlu vai tekstu. Dokumenta parakstītāja dati tiek glabāti atsevišķi anotācijas objektā.

Tomēr šie dati iekļauj sevī tikai lietotāja e-pastu. Visi paraksti ir redzami dokumentā un pievienoti dokumenta pēdējai lapai, kas arī ir ģenerēta uz servera. Šis risinājums balstās uz paraksta pieprasīšanas idejas, tātad lietotājam ir nepieciešams pieprasīt parakstu, ko kādas trešās personas izmanto tikai viņu e-pastu. "EchoSign" piedāvā lietojumprogrammas saskarni programmatūras izstrādātājiem. [21] Šī lietojumprogrammas saskarne galvenokārt piedāvā sekojošās funkcijas:

- Dokumenta aizsūtīšana uz pierakstu pa e-pastu;
- Dokumenta statusa pārbaude;
- Parakstīta dokumenta lejupielāde;
- Sīkrīka veidošana, kurš dod iespēju parakstīt dokumentu uz vietas;
- Sīkrīkā parakstīto dokumentu pārbaude;

Bastoties uz visu iegūto informāciju, var secināt, ka "EchoSign" pielietošanas scenārijs ir diezgan ierobežots. Galvenā problēma ir tāda, ka viss parakstu pieprasījums balstās uz e-pasta un pēc paša dokumenta ir diezgan grūti noteikt, kas ir to parakstījis. Tomēr to ir viegli lietot un nav nepieciešama nekāda papildus programmatūra.

"CoSign Client" ir elektronisko parakstu risinājums, kuru izstrādāja "ARX Inc.". Šis risinājums balstās uz klienta un servera mijiedarbības. PDF dokuments tiek atvērts, izmantojot "CoSign Client" lietotni, kad lietotājs grib parakstīt dokumentu, klients aizsūta pieprasījumu serverim. Serveris prasa klientu autentificēties, izmantojot lietotāja konta datus. Kad lietotājs ir autentificējies, dokumenta īssavilkums tiek aizsūtīts uz serveri un tur parakstīts, izmantojot lietotāja individuālo sertifikātu. Parakstīts īssavilkums tiek atgriezts klientam un pievienots dokumentam. Turklāt šis risinājums dod iespēju pievienot vairākus papildu datus parakstam. Ir iespēja pievienot arī rokas parakstu, izmantojot papildus ierīces. Klientu var uzkonfigurēt, lai izmantotu citus parakstu sniedzējus, un ir iespēja pieslēgt arī aparatūras drošības moduli. "CoSign" piedāvā arī bagātu lietojumprogrammas saskarni programmatūras izstrādātājiem. [22] Ar to iespējams ne tikai dokumentu parakstīt, bet strādāt arī ar sertifikātiem un dokumentiem. Neskatoties uz to, ka "CoSign" neatbalsta nekādas dokumenta maršrutēšanas iespējas, tai ir daudz vairāk iespēju priekš integrēšanas nekā "EchoSign". Tomēr "CoSign" risinājumam ir arī savi trūkumi, viens no tiem ir tas, ka visiem: gan dokumenta parakstītājam, gan dokumenta saņēmējam ir nepieciešams "CoSign Client", lai būtu iespēja dokumentu parakstīt un pārbaudīt. Šis risinājums var būt efektīvs uzņēmuma vai organizācijas sfērā.

"RightSignature" ir elektronisko parakstu risinājums, kuru izstrādāja "RightSignature LLC". Šis risinājums balstās uz klienta un servera mijiedarbības, tomēr tas neizmanto nekādas kriptogrāfiskās metodes tieši paraksta veidošanai. Dokumenti tiek augšupielādēti uz "RightSignatue" serveriem, kur tiem ir iespējams pievienot papildus laukus un aizpildīšanas

loģiku. Ir iespējams definēt kādai personai, kādus laukus ir nepieciešams aizpildīt. Nākamais solis ir definēt personu sarakstu, kas parakstīs šo dokumentu. Personas tiek definētas ievadot viņu vārdus un e-pasta adreses. Un dokumenti tiek aizsūtīti parakstītājiem. Parakstītājs nekad nesaņem pašu dokumentu, bet tikai saiti uz to. Dokuments tiek atvērs "RightSignature" tīmekļa lietotnē, kur lietotājs ievada datus un pievieno rokas parakstu. Pats paraksts tiek izveidots kā parasts attēls ar saiti un šo pašu dokumentu tīmekļa lietotnē. Pats dokuments neizmanto nekādas kriptogrāfiskās metodes integritātes pārbaudei. Integritāte tiek nodrošināta ar to, ka dokumentu var apskatīt uz servera un ir nepieciešams pilnīgi uzticēties parakstu sniedzējam. Sertifikāta vietā dokumentam tiek pievienotas papildus lapas ar informāciju par parakstītāju un pašu parakstīšanas transakciju. "RightSignature" piedāvā arī lietojumprogrammas saskarni, kas dod iespēju automatizēt dokumentu izsūtīšanu, ievietot dokumenta parakstīšanas sistēmu citā lietotnē un lejupielādēt dokumentus vai datus no dokumentiem. [23] Atkarībā no visiem iepriekš minētiem risinājumiem, šis risinājums neizmanto nekādus sertifikātu un drošības algoritmus tieši dokumenta pusē. Dokumenta integritāte var būt pārbaudīta, tikai izmantojot parakstu sniedzēju. Tātad, katru reizi, kad lietotājs saņem parakstītu dokumentu, ir nepieciešams vērsties pie paraksta sniedzēja un pārbaudīt dokumentu manuāli. Turklāt programmatūras izstrādātājam nav iespējams automatizēt dokumentu pārbaudes procesu, jo dokumenti nesatur nekādus sertifikātus.

Pēdējais apskatāmais risinājums būs "HelloSign" tīmekļa lietotne. Šis risinājums ir ļoti līdzīgs "RightSignature" risinājumam. Tas neizmanto sertifikātus dokumenta parakstīšanai. Pats parakstīšanas process pilnīgi sakrīt ar "RightSignature" pieeju, tomēr lielākā atšķirības ir tāda, ka parakstīšanas rezultātā tiek izveidots pilnīgi jauns dokuments, kas satur sevi vairāku dokumenta lapu attēlus. Izmantojot tikai dokumentu pilnīgi nekādā veidā nav iespējams pārbaudīt dokumenta integritāti, jo paša dokumenta attēlu ir iespējams izmainīt. Turklāt dokuments nesatur informāciju par parakstītāju, tikai parakstītāja paraksta attēlu. "HelloSign" piedāvā lietojumprogrammas saskarni, kura spēj tikai pievienot parakstu un lejupielādēt parakstītos dokumentus no "HelloSign" serveriem. [24]

4. ELEKTRONISKO PARAKSTU ĪSTENOŠANA

Lai pārliecinātos par to, ka ir iespējams uztaisīt lietotni, kas spēj parakstīt un pārbaudīt elektroniskos parakstus PDF dokumentā, vispirms ir nepieciešams atrast tehnoloģijas, kas mums dod iespēju strādāt ar PDF dokumentiem un to struktūru.

Priekš programmatūras izstrādātājiem eksistē daudz atsevišķu rīku un viena no visizplatītākām bibliotēkām darbam ar PDF dokumentiem ir iText. [25] Tā ir izstrādāta Java valodā un atbalsta visas 'Adobe System' atbalstītās manipulācijas. iText bibliotēka ir pārnesta uz .NET vidi. iTextSharp ir C# bibliotēka kas pēc piedāvātām iespējām pilnīgi atbilst Java versijai.

Tāpat kā iText, iTextSharp ir atvērta pirmkoda bibliotēka un tiek izplatīta pēc Affero GNU publiskās licences. Pēdējā iTextSharp versija pilnīgi atbilsts Java versijai un atbalsta dokumentu modificēšanu, veidošanu, parakstīšanu un cita veida apstrādes. Viena no šīs bibliotēkas priekšrocībām ir tāda, ka to atbalsta vairāki iText inženieri.

Neskatoties uz to, ka iTextSharp ir atvērtā koda bibliotēka, tā nav bezmaksas. Šobrīd bieži vien tiek izmantotas 2 dažādas iTextSharp versijas. Versija 5.5.2 ir jaunākā oficiālā versija un tā izmanto Affero GNU publisko licenci, kas nozīmē, ka, ja izstrādātājs grib izmantot to savā projektā, tam projektam arī vajag būt atvērtā kodā. Un situācijās, kad ir nepieciešams to izmantot aizvērta pirmkoda projektos, ir nepieciešams nopirkt licenci. Versija 4.1.6 ir pēdējā versija, kas bija licenzēta, izmantojot MPL & GLP licenci. Šī versija var būt izmantota jebkura veida projektos un neprasa nekādas papildus izmaksas, tomēr šī versija vairs nav oficiāli atbalstīta.

Tā kā mēs izmantosim iTextSharp bibliotēku darbam ar PDF dokumentiem ir nepieciešams atrast bibliotēku darbam ar sertifikātiem C# valodā. Šim nolūkam tika atrasta Bouncy Castle kriptogrāfiskā bibliotēka. [26] Tā galvenokārt tiks izmantota, lai ielādētu nepieciešamo informāciju no sertifikātiem. Turklāt šo bibliotēku atbalsta arī iTextSharp un tas atvieglos procesu.

Darba gaitā tika izveidota lietotne, kurai ir 2 galvenās funkcijas: pirmkārt, tā spēj parakstīt jebkuru PDF dokumentu, izmantojot PKCS#12 sertifikātu, un otrkārt, tā spēj pārbaudīt dokumentu un parakstu. Tā ir izstrādāta, izmantojot ASP.NET tehnoloģiju balsoties uz MVC 4 arhitektūru. Lietotnes galvenā ideja ir parādīt kā var viegli strādāt ar parakstiem, izmantojot izvēlētās tehnoloģijas.

4.1. PDF dokumenta parakstīšana, izmantojot iTextSharp

Lai parakstītu jebkuru PDF dokumentu, izstrādātājam ir nepieciešami dati par dokumentu, paraksta uzstādījumiem, sertifikātu un sertifikāta sniedzēju dati. Tika izveidota speciāla klase "SignatureModel" kas iekļauj sevī vairākus parakstam nepieciešamos parametrus (4.1. att.).

```
public class SignatureModel
{
    public string Reason {get;set;}
    public string FieldName { get; set;}
    public int X1{get; set;}
    public int Y1{ get; set;}
    public int X2{get; set;}
    public int Y2{ get; set;}
    public int PageNum{ get; set;}
    public float SigImageScal { get; set;}
    public string SigRenderMode{ get; set;}
    public string SigCertLevel{ get; set;}
    public string DigestAlg{ get; set;}
    public string CryptoStn { get; set; }
    public HttpPostedFileBase SigImage { get; set; }
    public string CrlURL { get; set; }
    public string TsaURL { get; set; }
    public string TsaUser { get; set; }
    public string TsaPass { get; set; }
    public string ValidationResults { get; set; }
}
```

4.1. att. **SignatureModel klases struktūra**

Galvenokārt šeit tiek glabātas kriptogrāfiskā standarta un īssavilkuma algoritma versijas. Visi pārējie parametri nav obligāti, bet galvenokārt tiek izmantoti paraksta izskata definēšanai un sertifikāta pārbaudei.

Vēl viena papildus klase tika izveidota darbam ar sertifikātiem. Tās nosaukums ir "CertificateModel" (4.2. att.).

```
public class CertificateModel
{
    public RsaPrivateCrtKeyParameters Key { get; set; }
    public ICollection<X509Certificate> Chain { get; set; }
}
```

4.2. att. **CertificateModel klases struktūra**

Šī klase glabā sevī informāciju par sertifikātu. Pirmkārt, tā ir sertifikāta privātā atslēga, ar kuras palīdzību tiks nodrošināta dokumenta integritāte. Otrkārt, tā ir sertifikātu ķēde, jo jebkurš sertifikāts var būt savienots ar vairākiem vecākiem sertifikātiem.

Lai pievienotu dokumenta parakstu, mums ir nepieciešams vispirms ielasīt sertifikāta privāto atslēgu un sertifikātu ķēdi. Tas tiek izdarīts, izmantojot iepriekš minēto "Bouncy Castle" bibliotēku (4.3. att.).

```
public CertificateModel PrepareCertificate(string KEYSTORE, char[] PASSWORD)
{
    Stream ksStream = new FileStream(KEYSTORE, FileMode.Open);
    Pkcs12Store store = new Pkcs12Store(ksStream, PASSWORD);
    string alias = "";
    ICollection<X509Certificate> chain = new List<X509Certificate>();
    foreach (string al in store.Aliases)
    {
        if (store.IsKeyEntry(al) && store.GetKey(al).Key.IsPrivate)
        {
            alias = al;
            break;
        }
    }
    AsymmetricKeyEntry pk = store.GetKey(alias);
    foreach (X509CertificateEntry c in store.GetCertificateChain(alias))
        chain.Add(c.Certificate);
    RsaPrivateCrtKeyParameters parameters = pk.Key as RsaPrivateCrtKeyParameters;
    CertificateModel result = new CertificateModel();
    result.Chain = chain;
    result.Key = parameters;
    ksStream.Close();
    return result;
}
```

4.3. att. **Datu ielasīšana no sertifikāta**

Kā var redzēt *4.3. att.* sertifikāts tiek ielādēts, izmantojot "Pkcs12Store" klasi. Šīs klases konstruktoram ir nepieciešami 2 galvenie parametri: sertifikāta plūsma. Šajā situācijā tā tiek ņemta no lokālā diska un sertifikāta atslēgas.

Nākamais solis ir sagatavoties darbam ar PDF dokumentu. Šim uzdevumam mēs esam izvēlējušies "iTextSharp" bibliotēku. Mums ir nepieciešams atvērt padoto dokumentu un sagatavot rakstīšanu jaunajā dokumentā (4.4. att.).

```
//creating PDF reader and stamper
PdfReader reader = new PdfReader(src);
FileStream outStream = new FileStream(dest, FileMode.Create);
PdfStamper stamper = PdfStamper.CreateSignature(reader, outStream, '\0');
```

4.4. att. **Sagatavošanās darbam ar PDF dokumentu**

Paņēmiens, kurš ir redzams 4.4. att. ir derīgs tikai tām situācijām, kad mēs strādājam ar jaunu dokumentu, kur nav neviena paraksta. Ja dokuments ir jau parakstīts, tad ir nepieciešams "PdfStamper" klases instanci izveidot tā saucamajā pievienošanas režīmā. Lai to izdarītu ir nepieciešams padot vienu papildus parametru. Tas ir nepieciešams tāpēc ka, jebkurš paraksts tiek pievienots visam dokumentam. Un, ja mēs ievietosim nākošo parakstu dokumentā, tad tas sabojās visus iepriekšējos parakstus. Lai izvairītos no šīs problēmas, tiek izmantots pievienošanas režīms, kas visas dokumenta izmaiņas raksta nevis pašā dokumentā, bet gan pievieno visas izmaiņas dokumenta beigās.

Kad sertifikāts un dokuments ir sagatavots, ir nepieciešams uzkonfigurēt parakstu (4.5. att.). Mēs varam izveidot jaunu parakstu jebkur dokumentā, vai nu parakstīt kādu iepriekš izveidotu paraksta lauku.

```
//creating signature appearance
PdfSignatureAppearance appearance = stamper.SignatureAppearance;
appearance.Reason = model.Reason;
appearance.SignDate = DateTime.Now;
appearance.Location = Request.ServerVariables["REMOTE_ADDR"];
//create new signature or sign existing field
if (string.IsNullOrEmpty(model.FieldName))
{
    appearance.SetVisibleSignature(new  iTextSharp.text.Rectangle(model.X1,  model.Y1,
model.X2, model.Y2),
    model.PageNum, "sig");
} else {
    appearance.SetVisibleSignature(model.FieldName);
}
```

4.5. att. **Paraksta konfigurēšana**

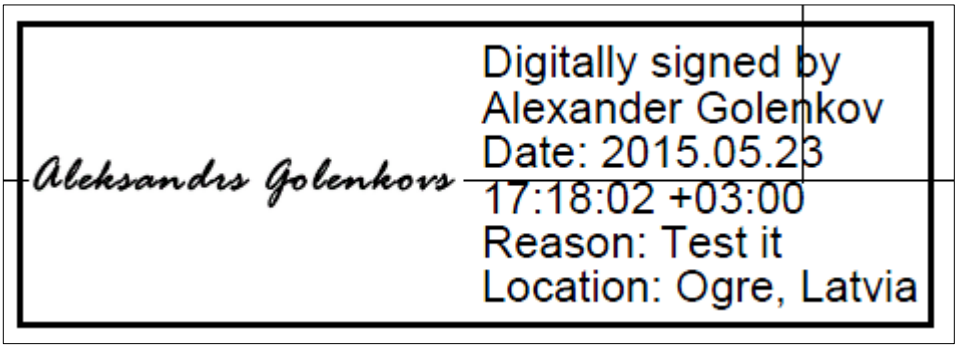
Neskatoties uz to, ka paraksts konfigurējas, izmantojot klasi "PdfSignatureAppearance", kurš ar savu nosaukumu aplami uzrāda, ka tā atbilst tikai paraksta izskatam, mēs varam pievienot arī papildus datus, kas uzglabāsies parakstā, sertifikātu atsaukšanas sarakstus un vairākus citus objektus.

Viena no ļoti populārām lietām šodien ir rokas paraksta attēla pievienošana parakstam. Ja mēs varam izmantojot mobilo telefonu vai citu ierīci konvertēt personas rokas parakstu digitālajā attēlā, tad mēs varam arī to pievienot mūsu parakstam (4.6. att.).

```
if (model.SigImage.ContentLength != 0){
    appearance.SignatureGraphic                                                    =
iTextSharp.text.Image.GetInstance(model.SigImage.InputStream);
    appearance.ImageScale = model.SigImageScal;
    PdfSignatureAppearance.RenderingMode rendMode = 0;
    switch (model.SigRenderMode){
        case "DESCRIPTION":
            rendMode = PdfSignatureAppearance.RenderingMode.DESCRIPTION;
            break;
        case "GRAPHIC":
            rendMode = PdfSignatureAppearance.RenderingMode.GRAPHIC;
            break;
        case "GRAPHIC_AND_DESCRIPTION":
            rendMode = PdfSignatureAppearance.RenderingMode.GRAPHIC_AND_DESCRIPTION;
            break;
        case "NAME_AND_DESCRIPTION":
            rendMode = PdfSignatureAppearance.RenderingMode.NAME_AND_DESCRIPTION;
        break;}
appearance.SignatureRenderingMode = rendMode;}
```

4.5. att. **Paraksta attēla pievienošana**

PDF dokumentu standarts atbalsta vairākus paraksta attēlošanas režīmus. Pēc noklusējuma paraksts tiek attēlots "apraksta" režīmā, kur tiek parādīta tikai bāzēs informācija par parakstu. Tomēr ir iespēja arī parādīt paraksta grafisko variantu (4.6. att.).



4.6. att. **Paraksta grafiskā režīma piemērs**

Turklāt "PDF standarts dod iespēju pievienot vairākus paraksta attēlošanas slāņus, kur var pievienot dažādus paraksta attēlus. Izmantojot šos attēlus un papildus datus par parakstu, var pievienot parakstam papildus drošības slāņus. Viens no piemēriem ir paņēmiens, kad ierīce rokas paraksta notveršanas ieraksta arī paraksta rokas spiedienu un citus fiziskos datus. Šos datus kopā ar paraksta attēlu var saglabāt paraksta metadatos.

Nākamais solis ir pievienot paraksta sertifikācijas līmeni, kas noteiks, kādas darbības ar PDF dokumentu ir atļautas pēc paraksta pievienošanas (4.7. att.).

```
int certLevel = 0;
switch (model.SigCertLevel){
      case "NOT_CERTIFIED":
               certLevel = PdfSignatureAppearance.NOT_CERTIFIED;
               break;
       case "CERTIFIED_FORM_FILLING":
               certLevel = PdfSignatureAppearance.CERTIFIED_FORM_FILLING;
               break;
       case "CERTIFIED_FORM_FILLING_AND_ANNOTATIONS":
               certLevel                                                                    =
PdfSignatureAppearance.CERTIFIED_FORM_FILLING_AND_ANNOTATIONS;
               break;
       case "CERTIFIED_NO_CHANGES_ALLOWED":
               certLevel = PdfSignatureAppearance.CERTIFIED_NO_CHANGES_ALLOWED;
               break;
}
appearance.CertificationLevel = certLevel;
```

4.7. att. **Paraksta sertifikācijas līmeņa definēšana**

Iepriekš jau tika aprakstīti visi iespējamie sertifikācijas līmeņi. iTextSharp bibliotēka pēc noklusējums pievieno visus parakstus ar sertifikācija līmeni 0, kas atbilst nesertificētam parakstam.

Īssavilkuma algoritms un kriptogrāfiskais standarts ir obligātie parametri, kas ir nepieciešami jebkuram parakstam (4.8. att.). Šie parametri tiks vēlāk izmantoti, lai izveidotu dokumenta jaucējsummu, kas ir nepieciešama dokumenta integritātes pārbaudīšanai.

```
string digestAlg = "";
switch (model.DigestAlg)
{
    case "RIPEMD160":
        digestAlg = DigestAlgorithms.RIPEMD160;
        break;
    case "SHA1":
        digestAlg = DigestAlgorithms.SHA1;
        break;
    case "SHA256":
        digestAlg = DigestAlgorithms.SHA256;
        break;
    case "SHA384":
        digestAlg = DigestAlgorithms.SHA384;
        break;
    case "SHA512":
        digestAlg = DigestAlgorithms.SHA512;
        break;
}
IExternalSignature pks = new PrivateKeySignature(pk, digestAlg);
CryptoStandard cryptoStd = 0;
switch (model.CryptoStn)
{
    case "CADES":
        cryptoStd = CryptoStandard.CADES;
        break;
    case "CMS":
        cryptoStd = CryptoStandard.CMS;
        break;
}
```

4.8. att. **Kriptogrāfiskā standarta un īssavilkuma algoritma izvēle**

iTextSharp bibliotēkā ir iebūvēti ne visi iespējamie algoritmi un standarti, tomēr tā piedāvā iespēju pievienot pašam savus algoritmus un standartus. Kā redzams *548. att.* 177. koda rindā, izmantojot privāto atslēgu, kuru mēs ieguvām no sertifikāta un īssavilkuma algoritmu, tiek izveidots "IExternalSignature" objekts. Tieši šis objekts nodarbojās ar PDF dokumenta jaucējsumma veidošanu un parakstīšanu.

Pēdējais solis pirms paraksta pievienošanas ir sertifikātu sniedzēju datu pievienošana. Šeit tiek iekļauti sertifikātu atsaukšanas saraksti, OCSP saites un TSA laikspiedolus (4.9. att.).

```
IList<ICrlClient> crlList = null;
if (!string.IsNullOrEmpty(model.CrlURL))
{
    crlList = new List<ICrlClient>();
    crlList.Add(new CrlClientOnline(model.CrlURL));
    crlList.Add(new CrlClientOnline(chain));
}
IOcspClient ocspClient = new OcspClientBouncyCastle();
ITSAClient tsaClient = null;
if (!string.IsNullOrEmpty(model.TsaURL) &&
    !string.IsNullOrEmpty(model.TsaUser) &&
    !string.IsNullOrEmpty(model.TsaPass))
{
    tsaClient = new TSAClientBouncyCastle(
        model.TsaURL, model.TsaUser, model.TsaPass
        );
}

MakeSignature.SignDetached(
    appearance, pks, chain,
    crlList, ocspClient,
    tsaClient, 0, cryptoStd);
//closing stamper
stamper.Close();
outStream.Close();
```

4.9. att. **Sertifikātu sniedzēju datu pievienošana un dokumenta parakstīšana**

Ir vairākas iespējas, kā var pievienot sertifikātu atsaukšanas sarakstu. Pirmkārt, var pašam ielasīt sarakstu no kāda faila. Otrkārt, var pievienot CRL tiešsaistes, kas tiks izmantotas saraksta lejupielādēšanai. Un pēdējais variants ir redzams *4.9. att.* koda 193. rindā. iTextSharp dod iespēju automātiski pievienot visus CRL tiešsaistes, kas jau ir minētas sertifikātu ķēdē. Atšķirībā no CRL sarakstiem, OCSP tiešsaistes var ielādēt tikai no sertifikātu ķēdes. Un pēdējais ir TSA laikspiedola pievienošana. Tiek izveidots klients, kas izmantojot akreditācijas datus automātiski, parakstīs mūsu parakstu ar TSA sertifikātu.

Kad visi paraksta dati tiek pievienoti, paraksts tiek pievienots dokumentam un visas plūsmas tiek aizvērtas.

4.2. Elektroniskā paraksta pārbaude, izmantojot iTextSharp

Pievienot PDF dokumenta parakstu ir vieglākā no parakstīšanas procesa daļām. Visas grūtības un problēmas rodas, kad mums ir nepieciešams pārbaudīt jau parakstīto PDF dokumentu.

Kā parastam lietotājam mums no parakstītā dokumenta ir vajadzīgi tikai sekojošie punkti:

- Dokumenta integritāte – pārbaudīt, vai dokuments nebija mainīts kopš paraksta pievienošanas.
- Paraksta uzticamība – pārliecināties, vai persona, kura saka, ka ir pievienojusi parakstu realitātē ir paraksta autors.

Kā izstrādātājiem mums ir vajadzīgi arī papildus dati no elektroniskā paraksta. Kad runa ir par dokumenta integritāti, iTextSharp piedāvā gatavas metodes PDF dokumentu pārbaudei. Lai pārliecinātos par to, ka pārbaudīt dokumentu ir ļoti viegli, tika uzrakstīta speciālā metode, kurai ir padots jebkurš PDF dokuments un kā rezultāts ir saņemts teksts ar informāciju par parakstu un tā uzstādījumiem.

Jebkurš paraksts PDF dokumentā ir pievienots kādam paraksta laukam. Šis lauks var būt neredzams gala lietotājam, tomēr PDF dokumentā tas vienmēr tiek uzglabāts. Pat ja dokumentā nav neviena paraksta lauka, pievienojot parakstu, ir nepieciešams to izveidot. Lai pārbaudītu dokumentu, mums ir nepieciešams iziet caur visiem parakstiem, kas ir pievienoti un pārbaudīt katru parakstu atsevišķi (4.10. att.).

```
PdfReader reader = new PdfReader(signedDoc);
AcroFields fields = reader.AcroFields;
List<String> names = fields.GetSignatureNames();
foreach (string name in names)
{
    result += "Checking signature: "+
        name + System.Environment.NewLine;
    //validate document integrity
    result += "1) Checking document integrity" +
        System.Environment.NewLine;
    result += "==== Signature covers whole document: " +
        fields.SignatureCoversWholeDocument(name) + System.Environment.NewLine;
    result += "==== Document revision: " +
        fields.GetRevision(name) + " of " +
        fields.TotalRevisions + System.Environment.NewLine;
    PdfPKCS7 pkcs7 = fields.VerifySignature(name);
    result += "==== Integrity: " +
        pkcs7.Verify() + System.Environment.NewLine;
    result += System.Environment.NewLine;
```

4.10. att. **Dokumenta integritātes pārbaude**

Lai izietu cauri visiem parakstiem PDF dokumentā, tiek izmantots "AcroFields" objekts, kas satur sevī visus dokumenta laukus. Turpmāk, izmantojot šo objektu, mēs varam piekļūt visai paraksta saglabātai informācijai. Kā redzams *4.10. att.*, no "AcroFields" klases objekta tiek ielasīts paraksta objekts, un vēlāk tas tiek pārbaudīts ar funkcijas "verify" palīdzību.

Izmantojot šādu paņēmienu, mēs iegūsim patiesu vērtību priekš katra dokumenta paraksta. Un šī patiesumvērtība parāda vai dokuments tika izmainīts pēc paraksta pievienošanas.

Mums paliek vēl 2 vietas, kuras mums ir nepieciešamas ielādēt no paraksta. Pirmkārt, tā ir informācija, kas tika pievienota pašam parakstam. Lai iegūtu šo informāciju, mums ir nepieciešams izmantot iepriekš izveidoto "PdfPKCS7" klases objektu. Šis objekts glabā sevī visu ISO-32000-1 standartā definētos parametrus.

```
result += "4) Reading signature metadata" + System.Environment.NewLine;
result += string.Format("==== Location: {0}", pkcs7.Location)
    + System.Environment.NewLine;
result += string.Format("==== Reason: {0}", pkcs7.Reason)
    + System.Environment.NewLine;
```

4.11. att. **Dokumenta paraksta datu lasīšanas piemērs**

Un ja nepieciešams ielasīt kādus papildus datus, kuri tika pievienoti paraksta pievienošanas procesā, tiem var piekļūt caur paraksta vārdnīcu.

Pēdējais, kas mums ir jāpārbauda, ir pats sertifikāts. Lai noteiktu vai sertifikāts ir derīgs un tam var ticēt, vajag pārbaudīt trīs lietas. Pirmkārt, ir nepieciešams pārbaudīt, vai pats sertifikāts ir derīgs un vai tas bija derīgs, kad dokuments tika parakstīts. Otrkārt, nepieciešams pārbaudīt vai tas neatrodas sertifikātu atsaukšanas sarakstā, ja tāds tiek atbalstīts. Un, treškārt, ir nepieciešams pārbaudīt vai mēs ticam šim sertifikātam.

Lai pārbaudītu vai sertifikāts ir derīgs, būs jāielasa visa sertifikātu ķēde no paraksta un jāpārbauda katra sertifikāta ķēde (4.12. att.).

```
X509Certificate[] certs = pkcs7.SignCertificateChain;
DateTime cal = pkcs7.SignDate;
for (int i = 0; i < certs.Length; ++i)
{
    result += "6) Checking certificate" + System.Environment.NewLine;
    result     +=     string.Format("====     Issuer:     {0}",     cert.IssuerDN)     +
System.Environment.NewLine;
    result     +=     string.Format("====     Subject:     {0}",     cert.SubjectDN)     +
System.Environment.NewLine;
    result += string.Format("==== Valid from: {0}", cert.NotBefore.ToString("dd/MM/yyyy
HH:mm:ss"))
        + System.Environment.NewLine;
    result  +=  string.Format("====  Valid  to:  {0}",  cert.NotAfter.ToString("dd/MM/yyyy
HH:mm:ss"))
        + System.Environment.NewLine;
```

4.12. att. **Paraksta sertifikātu ielasīšanas piemērs**

Katru sertifikātu ir nepieciešams pārbaudīt divas reizes. Galvenā pārbaude attiecās uz laiku, kad dokuments tika parakstīts, ir nepieciešams zināt vai tas bija derīgs šajā laika momentā. Un, ja tas bija derīgs, tad mēs varam pārbaudīt vai sertifikāts ir derīgs uz pašreizējo brīdi. Tomēr jāatceras, ka paraksts ir pilnīgi nederīgs, ja parakstīšanas laikā izmantojamais sertifikāts bija nederīgs.

```
try
{
```

```
    cert.CheckValidity(cal.ToLocalTime());
    result += string.Format("==== The certificate was valid at the time of signing.") +
System.Environment.NewLine;
}
catch (CertificateExpiredException e)
{
    result += string.Format("==== The certificate was expired at the time of signing.")
+ System.Environment.NewLine;
}
catch (CertificateNotYetValidException e)
{
    result += string.Format("==== The certificate wasn't valid yet at the time of
signing.") + System.Environment.NewLine;
}
try
{
    cert.CheckValidity();
    result += string.Format("==== The certificate is still valid.") +
System.Environment.NewLine;
}
catch (CertificateExpiredException e)
{
    result += string.Format("==== The certificate has expired.") +
System.Environment.NewLine;
}
catch (CertificateNotYetValidException e)
{
    result += string.Format("==== The certificate isn't valid yet.") +
System.Environment.NewLine;
}
```

4.13. att. **Paraksta sertifikātu ielasīšanas piemērs**

Mūsu izvēlētā bibliotēka iTextSharp piedāvā ļoti ērtu līdzekli, kas parāda vai sertifikāts ir derīgs parakstīšanas laikā un paraksta pārbaudes laikā. Katrai pārbaudei ir 3 iespējamie rezultāti: sertifikāts ir derīgs, sertifikāts jau ir nederīgs vai sertifikāts vēl nav derīgs.

Kad mēs esam pārliecinājušies par sertifikāta derīgumu, ir nepieciešams pārbaudīt, vai sertifikāts netika atsaukts pirms paraksta pievienošanas. Tas ir iespējams tikai tad, ja parakstīšanas procesa laikā tika pievienoti sertifikātu atsaukšanas saraksti vai tiešsaistes uz tiem. Lai pārbaudītu sertifikātu, ir nepieciešams uztaisīt pieprasījumu sertifikātu sniedzēju serverim un atkarībā no izvēlētās pieejas saņemt atbildi par sertifikāta derīgumu. Ja mēs plānojam pārbaudīt parakstus, kurus mēs nekontrolējam vajag pārbaudīt sertifikātu, izmantojot OPCS protokolu un CRL sarakstus.

```
List<BasicOcspResp> ocsps = new List<BasicOcspResp>();
if (pkcs7.Ocsp != null)
{
    ocsps.Add(pkcs7.Ocsp);
}
OcspVerifier ocspVerifier = new OcspVerifier(null, ocsps);
List<VerificationOK> verificationOnSign =
    ocspVerifier.Verify(signCert, issuerCert, cal);
if (verificationOnSign.Count == 0)
{
    List<X509Crl> crls = new List<X509Crl>();
    if (pkcs7.CRLs != null)
    {
        foreach (X509Crl crl in pkcs7.CRLs)
        {
            crls.Add(crl);
        }
    }
    CrlVerifier crlVerifier = new CrlVerifier(null, crls);
    verificationOnSign.AddRange(crlVerifier.Verify(signCert, issuerCert, cal));
}
if (verificationOnSign.Count == 0)
{
    result += string.Format("==== The signing certificate couldn't be verified") +
System.Environment.NewLine;
}
```

4.14. att. **Sertifikātu atsaukšanas pārbaude**

Visi sertifikāti tiek pārbaudīti 2 reizes, izmantojot laika momentu, kad dokuments tika parakstīts un pašreizējo laika momentu. Paraksts nav derīgs tikai tad, kad sertifikāts tika atsaukts pirms paraksta pievienošanas.

Kad mēs esam pārliecinājušies, ka paraksta sertifikāts ir derīgs, nepieciešams izdarīt pēdējo pārbaudi. Un šīs pārbaudes jautājums ir: vai mēs ticam šim parakstam un tieši sertifikātam, vai mēs esam pārliecināti, ka persona, kas ir minēta sertifikātā, ir reāls paraksta autors. Pēc savas būtības šī pārbaude ir viegla, mēs salīdzinām paraksta sertifikātu ar kādu sertifikātu bāzi, kurai mēs uzticamies.

```
List<X509Certificate> certificates = new List<X509Certificate>();
IList<VerificationException> errors = CertificateVerification.VerifyCertificates(certs,
certificates, null, cal);
if (errors == null){
    result += string.Format("==== Certificate verified") + System.Environment.NewLine;
} else {
    foreach (VerificationException error in errors)
    {
        result += error.Message + System.Environment.NewLine; ;
    }
}
```

4.15. att. **Sertifikātu uzticamības pārbaude**

Izstrādātā lietotne izmanto sertifikātus, kas iepriekš tika augšupielādēti uz servera. Balstoties uz šo nelielu sertifikātu kopu, katrs paraksts tiek salīdzināts ar sertifikātiem kopā, un, ja tur tāds sertifikāts jau ir, paraksts tiek uzskatīts par uzticamu. Tomēr lielākā problēma ir tieši šajā uzticamo sertifikātu bāzē.

4.3. Izstrādātā risinājuma problēmas

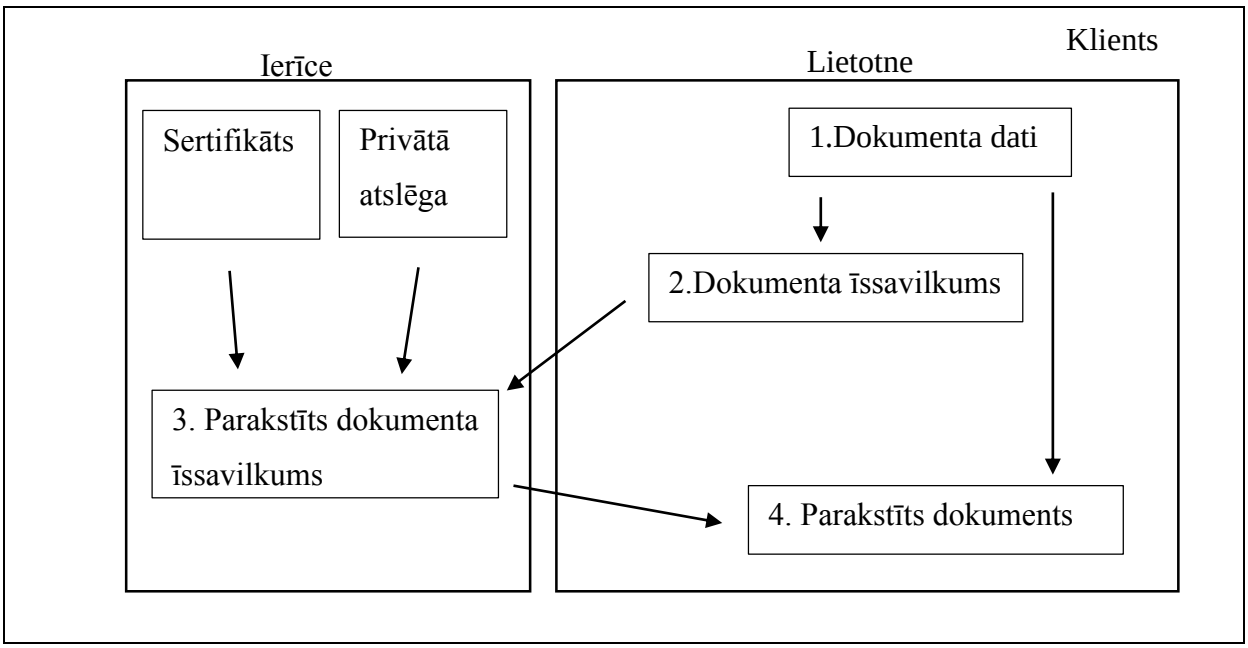
Tīmekļa lietotne, kura tika izstrādāta, salīdzina sertifikātus servera pusē. Tas nozīmē, ka tā ticēs tikai tiem sertifikātiem, kas bija speciāli augšupielādēti uz servera. Tad sanāk, ka izstrādāta lietotne uzticas tikai tiem parakstiem, kuriem uzticas servera administrators. Šī pieeja ir efektīva tikai tad, kad izstrādātājs ir pārliecināts, ka ir nepieciešams uzticēties tikai kādai konkrētai sertifikātu kopai. Turklāt, lai parakstītu dokumentu, lietotnei ir nepieciešams padot ne tikai sertifikātu, bet arī viņa atslēgu. Tas nozīmē, ka šos sensitīvos datus ir nepieciešams aizsūtīt uz serveri un tas atver lielu drošības caurumu. Kad sensitīvie dati tiek aizsūtīti uz serveri, ir iespēja, ka tos varēs izlasīt trešā persona, kurai šie dati nepienākas. Un šādā gadījumā sertifikātu vajadzēs vēlāk, pievienot atsaukto sertifikātu sarakstam. Šāda pieeja ir nepieļaujama reālajās sistēmās, jo privātā atslēga nekad nedrīkst tikt sūtīta.

Lietotnes izstrādes laikā, darba autoram radās vairākas problēmas ar izvēlēto "iTextSharp" bibliotēku. Viena no lielākām problēmām, strādājot ar šo bibliotēku ir tāda, ka tai nav oficiālās dokumentācijas. Vienīgie pieejamie materiāli ir uzrakstīti Java valodā iText bibliotēkai. Neskatoties uz to, ka Java un C# ir līdzīgas valodas, tomēr klašu nosaukumi un objektu tipi ļoti bieži nesakrita. Tas padarīja lietotnes izstrādi diezgan sarežģītu, jo bieži vien bija nepieciešams balstīties uz mēģinājumu un kļūdu metodi.

4.4. Izstrādātā risinājuma uzlabošanas iespējas

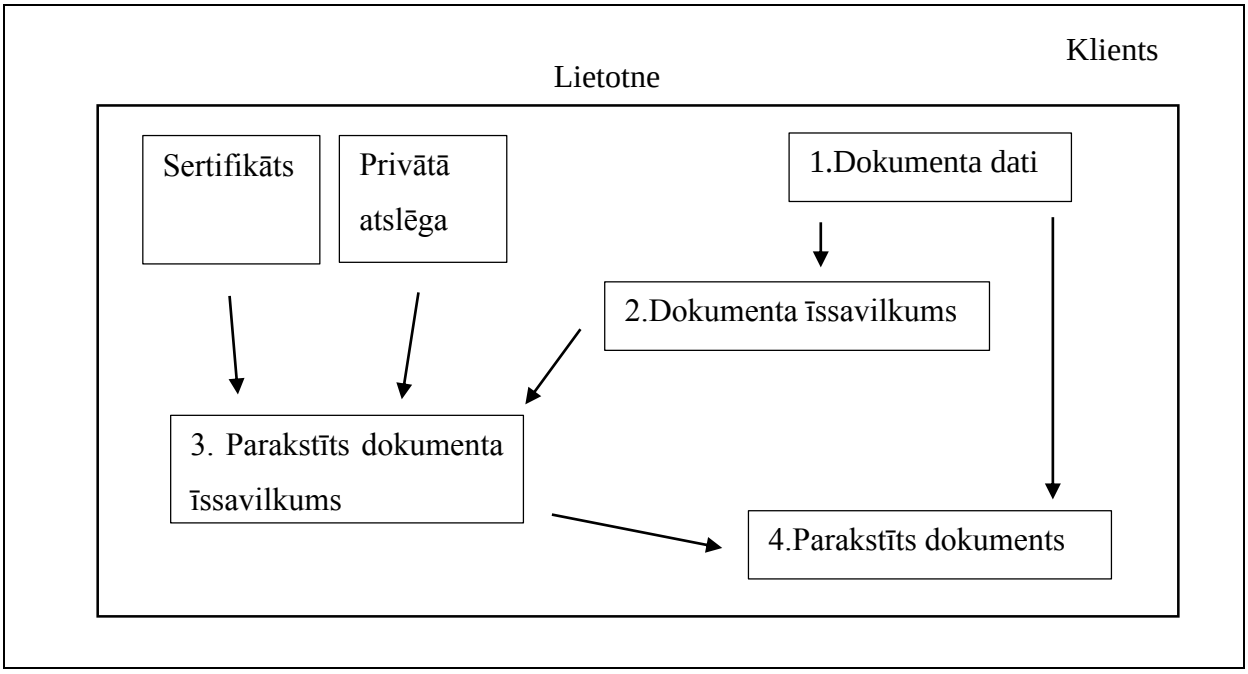
Iepriekš tika minēts, ka dažādi parakstu sniedzēji var izmantot dažādas pieejas. Realitātē pieeju priekš sertifikātu glabāšanas vajag izvēlēties atkarībā no konkrētās situācijas un problēmas. Galvenokārt visas pieejas dalās četrās grupās. Šo grupu galvenā atšķirība ir vieta, kur paraksts tiek izveidots un, kur dokuments tiek parakstīts: uz klienta vai uz servera.

Pirmie risinājumi ir izmainīt lietotni tā, lai paraksts tiktu izveidots un pievienots tieši klienta pusē. Šīs pieejas priekšrocība ir tāda, ka dokumentus būs iespēja parakstīt bez interneta pieslēguma un lietotājs ir pilnīgi pārliecināts par to, kādu dokumentu viņš ir parakstījis. Ir divas pieejas, kā var dokumentu parakstīt klienta pusē. Pirmā pieeja ir parādīta *att. 4.16.*, tā izmanto papildu ierīci priekš paraksta veidošanas. Šī pieeja var būt izmantota, kad ir nepieciešams parakstīt dokumentu, izmantojot kādu ārējo ierīci kā: zibatmiņu vai viedkarti. [27] [28]



4.16. att. **Klienta puses paraksts ar papildus ierīces palīdzību**

Otrā pieeja ir veidot parakstu un pievienot to tieši lietotnē. Pēc šī principa darbojās visi PDF dokumentu pārlūki, kas atbalsta parakstu pievienošanu. Daži piemēri ir: Adobe Reader un Foxit Reader. Kā var redzēt *att 4.17.* paraksta veidošana un pievienošana notiek tikai lietotnes ietvaros.



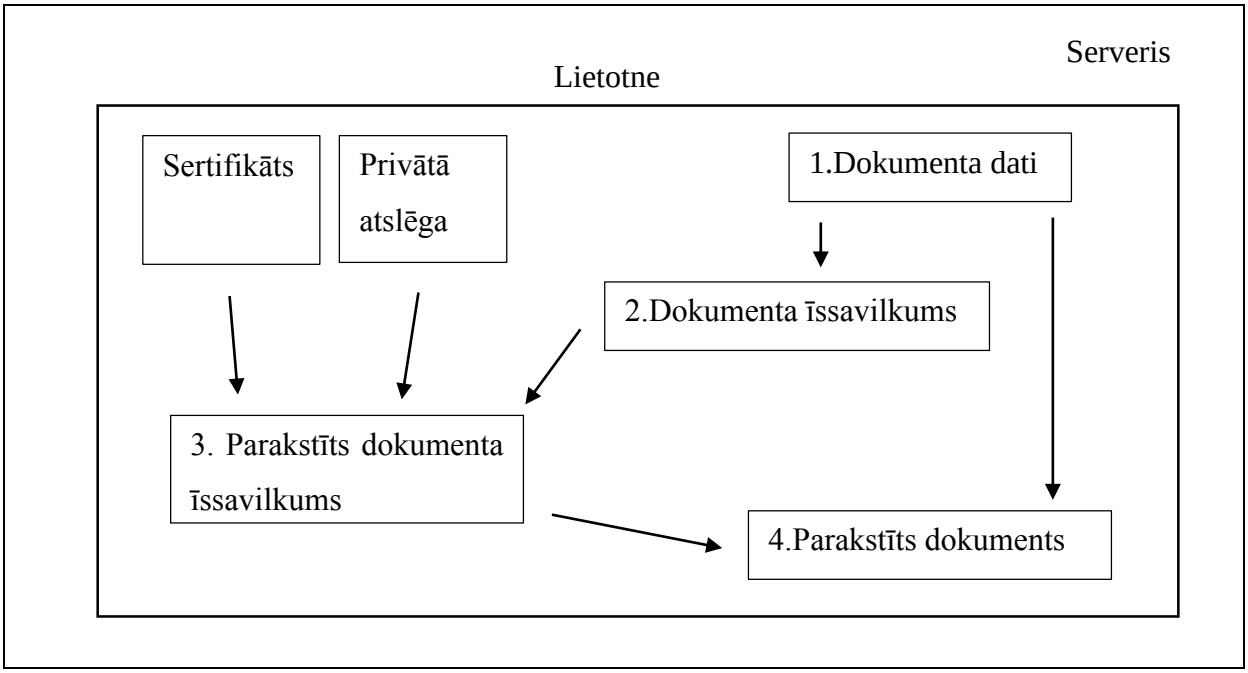
4.17. att. **Klienta puses paraksts bez papildus ierīcēm**

Tomēr ne visi risinājumi ir šodien tehniski realizējami. Viens no piemēriem ir situācija, kur ir nepieciešams pārbaudīt sertifikātu klienta pusē tīmekļa lietotnē. Izstrādātā tīmekļa lietotne spēs atdalīt sertifikātu no paraksta. Ir nepieciešams interneta pārlūkā pārbaudīt vai lietotājs uzticas šim sertifikātam. Tomēr tas nav iespējams, jo mūsdienās interneta pārlūki nedod iespēju piekļūt sertifikātu bāzei, kas tiek glabāta lietotāja ierīcē. Tas nozīmē, ka vienīgais

risinājums šai problēmai ir lejupielādēt sertifikātu un ar rokām pārbaudīt to, kas nav efektīvs risinājums.

Turklāt šim risinājumam ir daži lieli trūkumi, galvenais no tiem ir tas, ka mums ir nepieciešams uzticēties lietotāja lokālajam laikspiedolam. Kas nav droši, jo lietotājs var viegli mainīt savus lokālos uzstādījumus.

Otrais risinājums ir izveidot un pievienot parakstu tikai servera pusē. Šo pieeju ļoti bieži izmanto elektronisko parakstu sniedzēji, jo to ir vieglāk realizēt un tā ir drošāka nekā tīri klienta puses pievienotie paraksti (att. 4.18.). Viens no šādas sistēmas piemēriem ir "eparaksts.lv". Šādai sistēmai vajag strādāt uz parakstu sniedzēja serveriem un katram lietotājam ir nepieciešams apstiprināt autentiskumu pirms paraksta pievienošanas. Bieži vien visi parakstītie dokumenti glabāsies uz servera, lai būtu iespēja to pārbaudīt vēlāk.



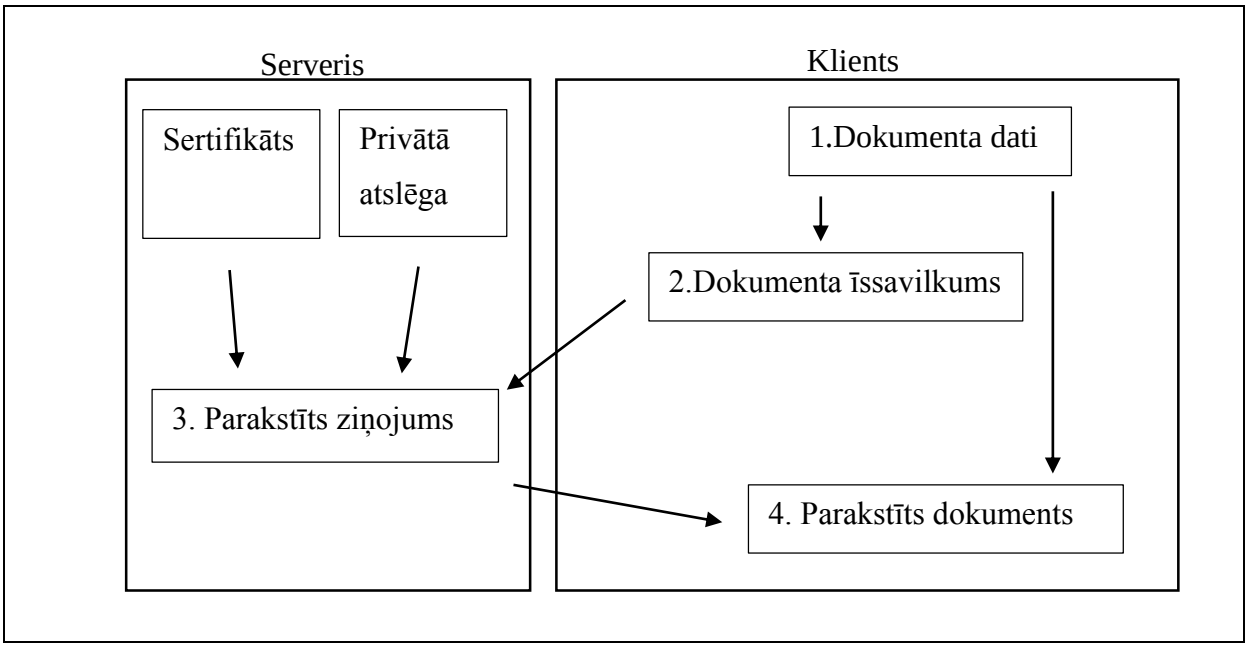
4.18. att. **Servera puses paraksts**

Šādam risinājumam ir iespējamas 2 sertifikātu glabāšanas pieejas. Pirmā pieeja ir parakstīt dokumentus, izmantojot paraksta sniedzēju sertifikātu un privāto atslēgu. Šajā situācijā lietotājs nav parakstītājs, bet drīzāk viņš dod atļauju parakstu sniedzējam parakstīties viņa vietā. Otrā pieeja ir ģenerēt katram lietotājam savu sertifikātu un parakstīt dokumentu, izmantojot tieši lietotāja sertifikātu. Tomēr, šī pieeja arī nav ideāla, jo šajā gadījumā parakstu sniedzējs ģenerē ne tikai sertifikātu, bet arī privāto atslēgu.

Izmantojot tīri servera risinājumu, ir nepieciešams ar katru parakstu saglabāt arī papildus datus kā: parakstītāja IP adrese, laikspiedols, lietotājvārdu un citus lietotāja konta datus. Tomēr tas nenodrošina to, ka tādiem parakstiem var pilnīgi ticēt. Pirmkārt, lietotājs nekad nav pārliecināts par to, kādu dokumentu viņš ir parakstījis, jo pats lietotājs redzēs tikai dokumentu kopijas, nevis pašus oriģinālos baitus. Otrkārt, mēs nevaram pārliecināties persona, kura ir

minēta kā paraksta autors, realitātē, parakstīja dokumentu. Mēs nevaram izslēgt situācijas, kad lietotāja lietotājvārds un parole kļūst kompromitēta. Turklāt var rasties arī situācijas, ka lietotāji ir ievadījuši aplamus lietotāja konta datus. Šo pieeju izmanto "Adobe EchoSign" risinājums. [29]

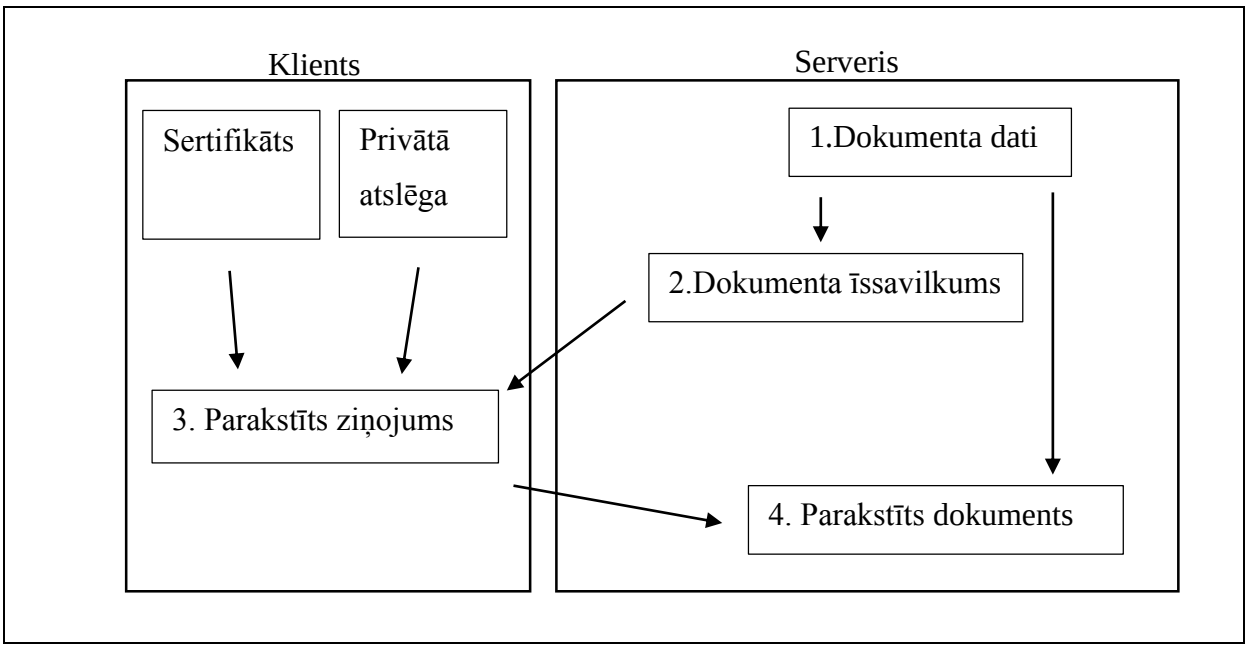
Trešais risinājums balstās uz servera un klienta mijiedarbības, izmantojot aparatūras drošības moduli. Šis risinājums bieži tiek izmantots kāda uzņēmuma var organizācijas ietvaros. Servera pusē tiek uzstādīts aparatūras drošības modulis, kas nodarbojas ar ziņojumu parakstīšanu. Drošības modulis var saņemt jebkurus datus, un viņš parakstīs tos, izmantojot savu sertifikātu un privāto atslēgu. Tālāk dati tiek atgriezti klienta pusē un pievienoti dokumentam (att. 4.19.).



4.19. att. **Lietotne klienta pusē un paraksts veidots servera pusē**

Šim risinājumam ir savi trūkumi, galvenokārt drošības modulis saņem tikai īssavilkumu un tas nozīmē, ka servera pusē nav iespējams uzglabāt pašus dokumentus. Tad dokumentu saglabāšana pāriet uz lietotāja atbildību, kas nav labākais risinājums. Šī pieeja tiek izmantota "CoSign Cloud" risinājumā. [30] [31]

Pēdējais risinājums ir saņemt dokumenta īssavilkumu no servera un izveidot parakstu klienta pusē. Šajā situācija dokuments tiek glabāts uz servera un, kad lietotājs izlemj pievienot parakstu, visa dokumenta vietā tiek atsūtīts tikai dokumenta īssavilkums. Bieži vien īssavilkums šajā situācijā tiek parakstīts, izmantojot ierīces kā zibatmiņu vai viedkarti.



4.20. att. **Lietotne servera pusē un paraksts veidots klienta pusē**

Tomēr šīs pieejas lielākā problēma ir tāda, ka lietotājs nevar pārliecināties, ko tieši viņš paraksta. Lietotājs saņem tikai dokumenta īssavilkumu, nevis pašu dokumentu.

Kā var redzēt, ir vairāki iespējami lietotnes uzlabošanas ceļi. Ja nākotnē būs nepieciešamība uzlabot šo lietotni, tad pirmkārt, ir nepieciešams cītīgi izanalizēt situāciju un drošības prasības un tikai tad izvēlēties vienu no iespējamiem risinājumiem. Katram no risinājumiem ir savas priekšrocības un savi trūkumi.

5. SECINĀJUMI

Šī darba ietvaros tika veikts elektronisko parakstu struktūras un funkcionalitātes pētījums. Vispirms tika izpētīts elektronisko parakstu koncepts un paraksta galvenie mērķi. Tika definēti vairāki pamatkritēriji, kuriem ir nepieciešams atbilst elektroniskam parakstam. Papildus tika apskatīti šodien pielietojamie parakstu tipi un to pielietojums.

Tālāk tika izpētīta elektroniskā paraksta struktūra un nepieciešamie dati elektronikā paraksta veidošanai. Tika apskatīti vairāki elektroniskā paraksta atribūti un to loma elektronikā paraksta veidošanas procesā. Līdz ar to tika aprakstīti un salīdzināti vairāki īssavilkuma un parakstīšanas algoritmi. Turklāt tika atrasti vairāki pētījumi, kas saistīti ar elektroniskiem parakstiem.

Lai pārliecinātos par to, ka eksistē vairāki risinājumi, elektronisko parakstu integrācijai, tika izpētīta elektronisko parakstu sniedzēju loma parakstīšanas procesā. Tika apskatīti vairākas drošības nodrošināšanas funkcijas, kuras nepieciešams īstenot jebkuram elektronisko parakstu sniedzējam. Papildus tika aprakstīti un salīdzināti vairāki eksistējošie elektronisko parakstu sniedzēji. Katrs risinājums tika pārbaudīts un tika aprakstīts risinājuma galvenā ideja, priekšrocības un trūkumi. Kā arī tika atrastas lietojumprogrammas, saskarnes katram risinājumam. Balstoties un iegūtiem rezultātiem var secināt, ka eksistē vairāki iespējamie risinājumi. Parastiem lietotājiem un arī programmatūras izstrādātājiem ir nepieciešams ļoti nopietni pieiet pie elektronisko parakstu sniedzēja izvēles, jo šodien neeksistē ideāla risinājuma elektronisko parakstu ieviešanai un katram risinājumam ir savas stiprās un vājās puses.

Darba ietvaros tika izstrādāta un aprakstīta tīmekļa lietotne, kuras uzdevums ir PDF dokumentu parakstīšana un parakstu pārbaudīšana. Papildus tika identificētas lietotnes drošības problēmas un tika piedāvāti vairāki problēmu risinājumi. Aprakstītie risinājumi pilnīgi atšķīrās pēc savas arhitektūras un paraksta pievienošanas pieejas. Katram risinājumam tika aprakstīts viņa pielietojums un problēmas. Balstoties uz iegūtiem rezultātiem var secināt, ka, izstrādājot elektronisko parakstu sistēmu, izstrādātājam ir nepieciešams identificēt sistēmas prasības un izvēlēties problēmas risinājumu atkarībā no konkrēta uzdevuma specifikas. Jo, kā jau tika iepriekš minēts, šodien neeksistē viens universāls risinājums.

Kopumā var secināt, ka elektroniskie paraksti kļūs vēl populārāki nākotnē un programmatūras izstrādātājiem jābūt gataviem parakstu integrēšanai savos produktos. Elektronisko parakstu integrēšana dod iespēju automatizēt ikdienas procesus un uzlabot uzņēmuma vai organizācijas produktivitāti.

AVOTU SARAKSTS

1. Elektronisko dokumentu likums. [tiešsaiste]. [atsauce 21.05.2015]. Pieejams Internetā: http://likumi.lv/ta/id/68521-elektronisko-dokumentu-likums
2. Usability of Biometrics in Relation to Electronic Signatures [tiešsaiste]. [atsauce 22.05.2015]. Pieejams Internetā: http://www.cse.lehigh.edu/prr/Biometrics/Archive/Papers/eubiosig.pdf
3. ISO 32000-1 standarts. [tiešsaiste]. [atsauce 21.05.2015]. Pieejams Internetā: http://www.iso.org/iso/catalogue_detail.htm?csnumber=51502
4. ISO 32000-2 standarts. [tiešsaiste]. [atsauce 21.05.2015]. Pieejams Internetā: http://www.iso.org/iso/catalogue_detail.htm?csnumber=63534
5. PDF dokumentu satura pārlūks. [tiešsaiste]. [atsauce 21.05.2015]. Pieejams Internetā: http://brendandahl.github.io/pdf.js.utils/browser/
6. PDF Reference. [tiešsaiste]. [atsauce 22.05.2015]. Pieejams Internetā: http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf
7. US Secure Hash Algorithm 1. [tiešsaiste]. [atsauce 22.05.2015]. Pieejams Internetā: http://www.hjp.at/doc/rfc/rfc3174.html
8. RSA Cryptography Specifications Version 2.1. [tiešsaiste]. [atsauce 22.05.2015]. Pieejams Internetā: https://tools.ietf.org/html/rfc3447
9. Johnson D., Menezes A., The Elliptic Curve Digital Signature Algorithm. [tiešsaiste]. [atsauce 22.05.2015]. Pieejams Internetā: http://residentrf.ucoz.ru/_ld/0/34_Digital_Signatu.pdf
10. SHA1 Deprecation Policy. [tiešsaiste]. [atsauce 23.05.2015]. Pieejams Internetā: http://blogs.technet.com/b/pki/archive/2013/11/12/sha1-deprecation-policy.aspx?Redirected=true
11. Barker E., Roginsky A., Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. [tiešsaiste]. [atsauce 23.05.2015]. Pieejams Internetā: http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf
12. Diffie W., Hellman M., New Directions in Cryptography. [tiešsaiste]. [atsauce 22.05.2015]. Pieejams Internetā: http://www.nku.edu/~christensen/092mat483%20DH%20paper.pdf
13. Malicious Hashing: Eve's Variant of SHA-1. [tiešsaiste]. [atsauce 24.05.2015]. Pieejams Internetā: http://eprint.iacr.org/2014/694.pdf

14. Hawkes P., Paddon M., Rose G. G., On Corrective Patterns for the SHA-2 Family. [tiešsaiste]. [atsauce 24.05.2015]. Pieejams Internetā: http://eprint.iacr.org/2004/207.pdf?q=secure-hash-standard-shs
15. Halevi S., Krawczyk H., Strengthening Digital Signatures via Randomized Hashing [tiešsaiste]. [atsauce 24.05.2015]. Pieejams Internetā: http://webee.technion.ac.il/~hugo/rhash/rhash.pdf
16. The Research of RSA-Based Undeniable Signature Method. [tiešsaiste]. [atsauce 25.05.2015]. Pieejams Internetā: http://link.springer.com/chapter/10.1007%2F978-3-642-31698-2_113
17. A New Generic Digital Signature Algorithm. [tiešsaiste]. [atsauce 25.05.2015]. Pieejams Internetā: http://www.uow.edu.au/~jennie/WEB/WEB11/2011_02.pdf
18. Roy A., Karforma S., A survey on digital signatures and its applications. [tiešsaiste]. [atsauce 25.05.2015]. Pieejams Internetā: http://www.academia.edu/2121568/A_survey_on_digital_signatures_and_its_applications
19. An investigation of the efficacy of electronic consenting interfaces of research permissions management system in a hospital setting. [tiešsaiste]. [atsauce 26.05.2015]. Pieejams Internetā: http://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC3779682&blobtype=pdf
20. Revocation checking and Chrome's CRL [tiešsaiste]. [atsauce 26.05.2015]. Pieejams Internetā: https://www.imperialviolet.org/2012/02/05/crlsets.html
21. EchoSign API [tiešsaiste]. [atsauce 26.05.2015]. Pieejams Internetā: https://secure.echosign.com/public/docs/EchoSignDocumentService20
22. Cosign API [tiešsaiste]. [atsauce 26.05.2015]. Pieejams Internetā: http://docs.arx.com/CoSign_APIs/doc_v7.2/Default.htm#doc_7.2/doc.htm#_Toc407700291%3FTocPath%3D_____1
23. RightSignature API [tiešsaiste]. [atsauce 26.05.2015]. Pieejams Internetā: https://rightsignature.com/esignature-api
24. HelloSign API [tiešsaiste]. [atsauce 26.05.2015]. Pieejams Internetā: https://www.hellosign.com/api/documentation
25. Lowagie B., iText in Action, Second Edition – Manning Publications Co. 2010.
26. Bouncy Castle documentation [tiešsaiste]. [atsauce 27.05.2015]. Pieejams Internetā: https://www.bouncycastle.org/documentation.html

27. Architecture of ID-software [tiešsaiste]. [atsauce 27.05.2015]. Pieejams Internetā: https://open-eid.github.io/#_Toc403732160
28. An extensible client platform for eID, signatures and more [tiešsaiste]. [atsauce 27.05.2015]. Pieejams Internetā: http://www.ecsec.de/fileadmin/Ecsec-files/pub//2013_OID_Platform.pdf
29. Adobe EchoSign Security [tiešsaiste]. [atsauce 27.05.2015]. Pieejams Internetā: http://e-signatures.com.au/wp-content/uploads/2014/09/EchoSign-Security-Overview.pdf
30. CoSign Cloud – Fact Sheet [tiešsaiste]. [atsauce 28.05.2015]. Pieejams Internetā: http://www.arx.com/files/DOCUMENTS/CS-Cloud-fact-sheet.pdf
31. A better Approach to PKI based Digital Signatures [tiešsaiste]. [atsauce 28.05.2015]. Pieejams Internetā: http://www.cryptotech.com.pl/compl/data/arx/pdf/PKI_Digital_Signatures_White_Paper.pdf

PIELIKUMS 1. IZSTRĀDĀTĀS LIETOTNES PIRMKODS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

using System.IO;
using System.util;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Pkcs;
using Org.BouncyCastle.Tsp;
using Org.BouncyCastle.X509;
using Org.BouncyCastle.Security.Certificates;
using Org.BouncyCastle.Ocsp;
using Org.BouncyCastle.Crypto.Parameters;
using iTextSharp.text;
using iTextSharp.text.pdf;
using iTextSharp.text.pdf.security;
using SignApp.Models;
using System.Drawing;



namespace SignApp.Controllers
{
    public class HomeController : Controller
    {
        string src = @"..\Signature Test_clean.pdf";
        string srcfield = @"..\Signature Test_SigField.pdf";
        string outPut = @"..\SignedPDF{0}.pdf";
        string imageSRC = @"..\sigImg.PNG";
        string signedDoc = @"..\Signature Test_clean_EchoSign.pdf";

        string KEYSTORE = @"..\AlexanderGolenkov.pfx";
        char[] PASSWORD = "password".ToCharArray();

        public ActionResult Index()
        {

            SignatureModel mod = new SignatureModel();
            if (!string.IsNullOrEmpty(Request["ValidationResults"])) mod.ValidationResults =
Request["ValidationResults"].Replace(System.Environment.NewLine," <br />");
            return View(mod);
        }



        [HttpPost]
        public ActionResult SignForm(SignatureModel model)
        {
            //Stream file = Request.Files["SigImage"].InputStream;
            CertificateModel cert = PrepareCertificate(KEYSTORE,PASSWORD);
            sign(src,      string.Format(outPut,      DateTime.Now.TimeOfDay.TotalMinutes),
cert.Chain, cert.Key, model);
            return RedirectToAction("Index", "Home");
        }
        public ActionResult ValidateForm(SignatureModel model)
        {

            model.ValidationResults = validateSignature();
            return RedirectToAction("Index", "Home", model);
        }
```

```
public CertificateModel PrepareCertificate(string KEYSTORE, char[] PASSWORD)
{
    Stream ksStream = new FileStream(KEYSTORE, FileMode.Open);
    Pkcs12Store store = new Pkcs12Store(ksStream, PASSWORD);
    string alias = "";
    ICollection<X509Certificate> chain = new List<X509Certificate>();
    foreach (string al in store.Aliases)
    {
        if (store.IsKeyEntry(al) && store.GetKey(al).Key.IsPrivate)
        {
            alias = al;
            break;
        }
    }
    AsymmetricKeyEntry pk = store.GetKey(alias);
    foreach (X509CertificateEntry c in store.GetCertificateChain(alias))
        chain.Add(c.Certificate);
    RsaPrivateCrtKeyParameters parameters = pk.Key as RsaPrivateCrtKeyParameters;
    CertificateModel result = new CertificateModel();
    result.Chain = chain;
    result.Key = parameters;
    ksStream.Close();
    return result;

}

public void sign(string src, string dest,
    ICollection<X509Certificate> chain,
    ICipherParameters pk, SignatureModel model)
{
    //creating PDF reader and stamper
    PdfReader reader = new PdfReader(src);
    FileStream outStream = new FileStream(dest, FileMode.Create);
    PdfStamper stamper = PdfStamper.CreateSignature(reader, outStream, '\0');
    //creating signature appearance
    PdfSignatureAppearance appearance = stamper.SignatureAppearance;
    appearance.Reason = model.Reason;
    appearance.SignDate = DateTime.Now;
    appearance.Location = Request.ServerVariables["REMOTE_ADDR"];
    //create new signature or sign existing field
    if (string.IsNullOrEmpty(model.FieldName))
    {
        appearance.SetVisibleSignature(
            new iTextSharp.text.Rectangle(model.X1, model.Y1, model.X2, model.Y2),
            model.PageNum, "sig");
    }
    else
    {
        appearance.SetVisibleSignature(model.FieldName);
    }
    //set signature image
    if (model.SigImage.ContentLength != 0)
    {
        appearance.SignatureGraphic =
            iTextSharp.text.Image.GetInstance(model.SigImage.InputStream);
        appearance.ImageScale = model.SigImageScal;
        PdfSignatureAppearance.RenderingMode rendMode = 0;
        switch (model.SigRenderMode)
        {
            case "DESCRIPTION":
                rendMode = PdfSignatureAppearance.RenderingMode.DESCRIPTION;
                break;
            case "GRAPHIC":
                rendMode = PdfSignatureAppearance.RenderingMode.GRAPHIC;
                break;
            case "GRAPHIC_AND_DESCRIPTION":
```

```
                        rendMode                                                    =
PdfSignatureAppearance.RenderingMode.GRAPHIC_AND_DESCRIPTION;
                        break;
                    case "NAME_AND_DESCRIPTION":
                        rendMode                                                    =
PdfSignatureAppearance.RenderingMode.NAME_AND_DESCRIPTION;
                        break;
                }
                appearance.SignatureRenderingMode = rendMode;
            }

            //set certification level
            int certLevel = 0;
            switch (model.SigCertLevel)
            {
                case "NOT_CERTIFIED":
                    certLevel = PdfSignatureAppearance.NOT_CERTIFIED;
                    break;
                case "CERTIFIED_FORM_FILLING":
                    certLevel = PdfSignatureAppearance.CERTIFIED_FORM_FILLING;
                    break;
                case "CERTIFIED_FORM_FILLING_AND_ANNOTATIONS":
                    certLevel                                                   =
PdfSignatureAppearance.CERTIFIED_FORM_FILLING_AND_ANNOTATIONS;
                    break;
                case "CERTIFIED_NO_CHANGES_ALLOWED":
                    certLevel = PdfSignatureAppearance.CERTIFIED_NO_CHANGES_ALLOWED;
                    break;
            }
            appearance.CertificationLevel = certLevel;
            //creating signature
            string digestAlg = "";
            switch (model.DigestAlg)
            {
                case "RIPEMD160":
                    digestAlg = DigestAlgorithms.RIPEMD160;
                    break;
                case "SHA1":
                    digestAlg = DigestAlgorithms.SHA1;
                    break;
                case "SHA256":
                    digestAlg = DigestAlgorithms.SHA256;
                    break;
                case "SHA384":
                    digestAlg = DigestAlgorithms.SHA384;
                    break;
                case "SHA512":
                    digestAlg = DigestAlgorithms.SHA512;
                    break;
            }
            IExternalSignature pks = new PrivateKeySignature(pk, digestAlg);
            CryptoStandard cryptoStd = 0;
            switch (model.CryptoStn)
            {
                case "CADES":
                    cryptoStd = CryptoStandard.CADES;
                    break;
                case "CMS":
                    cryptoStd = CryptoStandard.CMS;
                    break;
            }
            IList<ICrlClient> crlList = null;
            if (!string.IsNullOrEmpty(model.CrlURL))
            {
                crlList = new List<ICrlClient>();
                crlList.Add(new CrlClientOnline(model.CrlURL));
                crlList.Add(new CrlClientOnline(chain));
            }
```

```
            IOcspClient ocspClient = new OcspClientBouncyCastle();
            ITSAClient tsaClient = null;
            if (!string.IsNullOrEmpty(model.TsaURL) &&
                !string.IsNullOrEmpty(model.TsaUser) &&
                !string.IsNullOrEmpty(model.TsaPass))
            {
                tsaClient = new TSAClientBouncyCastle(
                    model.TsaURL, model.TsaUser, model.TsaPass
                    );
            }

            MakeSignature.SignDetached(
                appearance, pks, chain,
                crlList, ocspClient,
                tsaClient, 0, cryptoStd);
            //closing stamper
            stamper.Close();
            outStream.Close();


        }

        public string validateSignature()
        {
            string result = "";

            PdfReader reader = new PdfReader(signedDoc);
            AcroFields fields = reader.AcroFields;
            List<String> names = fields.GetSignatureNames();

            foreach (string name in names)
            {
                result += "Checking signature: "+
                    name + System.Environment.NewLine;
                //validate document integrity
                result += "1) Checking document integrity" +
                    System.Environment.NewLine;
                result += "==== Signature covers whole document: " +
                    fields.SignatureCoversWholeDocument(name) + System.Environment.NewLine;
                result += "==== Document revision: " +
                    fields.GetRevision(name) + " of " +
                    fields.TotalRevisions + System.Environment.NewLine;
                PdfPKCS7 pkcs7 = fields.VerifySignature(name);
                result += "==== Integrity: " +
                    pkcs7.Verify() + System.Environment.NewLine;
                result += System.Environment.NewLine;

                //checking signature data
                result += "2) Checking signature properties" + System.Environment.NewLine;
                IList<AcroFields.FieldPosition> fieldPosList =
fields.GetFieldPositions(name);
                if (fieldPosList != null && fieldPosList.Count > 0)
                {
                    AcroFields.FieldPosition fieldPos = fieldPosList[0];
                    iTextSharp.text.Rectangle pos = fieldPos.position;
                    if (pos.Width == 0 || pos.Height == 0)
                    {
                        result += "==== Invisible signature "  + System.Environment.NewLine;
                    }
                    else
                    {
                        result += string.Format("==== Field on page {0}; x1: {1}, y1: {2},
x2:  {3};  y2:  {4}  ",fieldPos.page,  pos.Left,  pos.Bottom,  pos.Right,  pos.Top)  +
System.Environment.NewLine;
                    }
                }
                result       +=       string.Format("====       Digest       algorithm:
{0}",pkcs7.GetHashAlgorithm()) + System.Environment.NewLine;
```

```
                result += string.Format("==== Encryption algorithm: {0}",
pkcs7.GetEncryptionAlgorithm()) + System.Environment.NewLine;
                result += string.Format("==== Filter subtype: {0}", pkcs7.GetFilterSubtype())
+ System.Environment.NewLine;
                result += System.Environment.NewLine;
                result += "3) Checking certificate properties" + System.Environment.NewLine;
                X509Certificate cert = pkcs7.SigningCertificate;
                result += string.Format("==== Signer: {0}",
CertificateInfo.GetSubjectFields(cert).GetField("CN")) + System.Environment.NewLine;

                if (pkcs7.SignName != null)
                {
                    result += string.Format("==== Signer alternative name: {0}",
pkcs7.SignName) + System.Environment.NewLine;
                }

                result += string.Format("==== Signed on: {0}",
pkcs7.SignDate.ToString("MM/dd/yyyy HH:mm:ss")) + System.Environment.NewLine;
                if (!pkcs7.TimeStampDate.Equals(DateTime.MaxValue))
                {
                    result += string.Format("==== Timestamp: {0}",
pkcs7.TimeStampDate.ToString("MM/dd/yyyy HH:mm:ss")) + System.Environment.NewLine;
                    TimeStampToken ts = pkcs7.TimeStampToken;
                    result += string.Format("==== TSA: {0}", ts.TimeStampInfo.Tsa) +
System.Environment.NewLine;
                    result += string.Format("==== Timestamp verified: {0}",
pkcs7.VerifyTimestampImprint()) + System.Environment.NewLine;
                }
                result += System.Environment.NewLine;
                result += "4) Reading signature metadata"
                    + System.Environment.NewLine;
                result += string.Format("==== Location: {0}", pkcs7.Location)
                    + System.Environment.NewLine;
                result += string.Format("==== Reason: {0}", pkcs7.Reason)
                    + System.Environment.NewLine;

                PdfDictionary sigDict = fields.GetSignatureDictionary(name);
                result += System.Environment.NewLine;
                result += "5) Checking segnature permissions" + System.Environment.NewLine;
                SignaturePermissions perms = new SignaturePermissions(sigDict, null);
                result += string.Format("==== Signature type: {0}", (perms.Certification ?
"Certification" : "Approval")) + System.Environment.NewLine;
                result += string.Format("==== Filling out fields allowed: {0}",
perms.FillInAllowed) + System.Environment.NewLine;
                result += string.Format("==== Adding annotations allowed: {0}",
perms.AnnotationsAllowed) + System.Environment.NewLine;

                foreach (SignaturePermissions.FieldLock Lock in perms.FieldLocks)
                {

                    result += string.Format("==== Locked field: {0}", Lock) +
System.Environment.NewLine;
                }

                X509Certificate[] certs = pkcs7.SignCertificateChain;
                DateTime cal = pkcs7.SignDate;
                for (int i = 0; i < certs.Length; ++i)
                {
                    result += "6) Checking certificate" + System.Environment.NewLine;
                    result += string.Format("==== Issuer: {0}", cert.IssuerDN) +
System.Environment.NewLine;
                    result += string.Format("==== Subject: {0}", cert.SubjectDN) +
System.Environment.NewLine;
                    result += string.Format("==== Valid from: {0}",
cert.NotBefore.ToString("dd/MM/yyyy HH:mm:ss"))
                        + System.Environment.NewLine;
                    result += string.Format("==== Valid to: {0}",
cert.NotAfter.ToString("dd/MM/yyyy HH:mm:ss"))
```

```
                    + System.Environment.NewLine;

                try
                {
                    cert.CheckValidity(cal.ToLocalTime());
                    result += string.Format("==== The certificate was valid at the time
of signing.") + System.Environment.NewLine;
                }
                catch (CertificateExpiredException e)
                {
                    result += string.Format("==== The certificate was expired at the time
of signing.") + System.Environment.NewLine;
                }
                catch (CertificateNotYetValidException e)
                {
                    result += string.Format("==== The certificate wasn't valid yet at the
time of signing.") + System.Environment.NewLine;
                }
                try
                {
                    cert.CheckValidity();
                    result += string.Format("==== The certificate is still valid.") +
System.Environment.NewLine;
                }
                catch (CertificateExpiredException e)
                {
                    result += string.Format("==== The certificate has expired.") +
System.Environment.NewLine;
                }
                catch (CertificateNotYetValidException e)
                {
                    result += string.Format("==== The certificate isn't valid yet.") +
System.Environment.NewLine;
                }
            }


            X509Certificate signCert = certs[0];
            X509Certificate issuerCert = (certs.Length > 1 ? certs[1] : null);
            result += string.Format("==== Checking validity of the certificate at the
time of signing") + System.Environment.NewLine;
            List<BasicOcspResp> ocsps = new List<BasicOcspResp>();
            if (pkcs7.Ocsp != null)
            {
                ocsps.Add(pkcs7.Ocsp);
            }
            OcspVerifier ocspVerifier = new OcspVerifier(null, ocsps);
            List<VerificationOK> verificationOnSign =
                ocspVerifier.Verify(signCert, issuerCert, cal);
            if (verificationOnSign.Count == 0)
            {
                List<X509Crl> crls = new List<X509Crl>();
                if (pkcs7.CRLs != null)
                {
                    foreach (X509Crl crl in pkcs7.CRLs)
                    {
                        crls.Add(crl);
                    }
                }
                CrlVerifier crlVerifier = new CrlVerifier(null, crls);
                verificationOnSign.AddRange(crlVerifier.Verify(signCert,     issuerCert,
cal));
            }
            if (verificationOnSign.Count == 0)
            {
                result += string.Format("==== The signing certificate couldn't be
verified") + System.Environment.NewLine;
            }
```

```
                else
                {
                    foreach (VerificationOK v in verificationOnSign)
                    {
                        result += string.Format(v.ToString()) + System.Environment.NewLine;
                    }
                }
                result += string.Format("==== Checking validity of the certificate today") +
System.Environment.NewLine;
                List<VerificationOK> verificationNow =
                    ocspVerifier.Verify(signCert, issuerCert, DateTime.Now);
                if (verificationNow.Count == 0)
                {
                    List<X509Crl> crls = new List<X509Crl>();
                    if (pkcs7.CRLs != null)
                    {
                        foreach (X509Crl crl in pkcs7.CRLs)
                        {
                            crls.Add(crl);
                        }
                    }
                    CrlVerifier crlVerifier = new CrlVerifier(null, crls);
                    verificationNow.AddRange(crlVerifier.Verify(signCert,       issuerCert,
DateTime.Now));
                }
                if (verificationOnSign.Count == 0)
                {
                    result  +=  string.Format("====  The  signing  certificate  couldn't  be
verified, because no CRL or OCSP provided") + System.Environment.NewLine;
                }
                else
                {
                    foreach (VerificationOK v in verificationNow)
                    {
                        Console.WriteLine(v);
                    }
                }

                List<X509Certificate> certificates = new List<X509Certificate>();
                IList<VerificationException>                    errors                    =
CertificateVerification.VerifyCertificates
                    (certs, certificates, null, cal);
                if (errors == null)
                {
                    result    +=    string.Format("====    Certificate    verified")    +
System.Environment.NewLine;
                }
                else
                {
                    foreach (VerificationException error in errors)
                    {
                        result += error.Message + System.Environment.NewLine; ;
                    }
                }



                result += System.Environment.NewLine;
            }

            return result;
        }

    }
}
```

Bakalaura darbs „Elektronisko parakstu īstenošana PDF dokumentos” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: ..... Aleksandrs Goļenkovs

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: M. dat. Artūrs Lavrenovs ..... 01.06.2015.

Recenzents: docents Dr.sc.comp. Aivars Niedrītis

Darbs iesniegts Datorikas fakultātē 01.06.2015.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe .....

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

..... prot. Nr. .....

Komisijas sekretāre: .....