Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Elektronický podpis v informačných systémoch Univerzity Komenského

Bakalárska práca

2018
Lukáš Kiss

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Elektronický podpis v informačných systémoch Univerzity Komenského

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. RNDr. Daniel Olejár, PhD.
Konzultant:	Mgr. Peter Kopáč

Bratislava, 2018
Lukáš Kiss

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lukáš Kiss
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Elektronický podpis v informačných systémoch Univerzity Komenského
Electronic signature in information systems of Comenius University

Anotácia: Analyzovať možnosti využívania elektronického podpisu v existujúcich informačných systémoch Univerzity Komenského, navrhnúť a implementovať/ integrovať riešenia na vytváranie a overovanie elektronických podpisov pomocou občianskych preukazov do systémov UK. V súvislosti s odhalenými slabinami OP analyzovať bezpečnosť navrhovaného riešenia a dôsledky zmien kľúčov, algoritmov v občianskych preukazoch pre navrhované riešenie.

Kľúčové slová: elektronický podpis, e-gov, informačné systémy UK

Vedúci: doc. RNDr. Daniel Olejár, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 21.11.2017

Dátum schválenia: 10.01.2018 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Týmto by som chcel poďakovať všetkým, čo mi pomohli pri riešení bakalárskej práce, hlavne svojmu vedúcemu **doc. RNDr. Danielovi Olejárovi, PhD.**, ktorý ma uviedol do problematiky a viedol ma počas celej bakalárskej práce. Za ochotu a ústretovosť ďakujem **Mgr. Peterovi Kopáčovi**, ktorý mi pomohol pri návrhu a programovaní demo riešenia. Nakoniec by som chcel poďakovať **Mgr. Matejovi Zagibovi**, ktorý mi pohol pri ladení a hľadaní chýb v demo riešení.

Abstrakt

Táto bakalárska práca sa zaoberá využitím občianskeho preukazu s čipom na autentifikáciu a vytváranie elektronického podpisu a popisuje návrh skladajúci sa z webového rozšírenia a lokálnej aplikácie. V prvej časti popisuje ich komunikáciu cez post požiadavky a následne opisuje demo verziu tohto návrhu, ktorá dokáže vytvoriť elektronický podpis, získať certifikát a overiť elektronický podpis pomocou certifikátu. Kvôli interakcii mnoho modulov, v predposlednej kapitole táto práca popisuje automatickú inštaláciu demo riešenia. Pri jej možnom zlyhaní, vysvetľuje aj manuálnu inštaláciu ako pod OS Windows, tak aj pod OS Linux. V poslednej časti sa práca venuje skúmaniu možných vylepšení demo riešenia a analyzuje aj jeho bezpečnosť.

Kľúčové slová: elektronický podpis, občiansky preukaz s čipom, webové rozšírenie

Abstract

This bachelor thesis deals with the use of a citizen card with an electronic chip and the creation of an electronic signature and describes a proposal consisting of web extension and local application. In the first part, it describes their communication via POST requests and then describes a demo version of this design that can create an electronic signature, obtain a certificate and verify the electronic signature with a certificate. Due to the interaction of many modules, in the penultimate chapter this work describes the automatic installation of a demo solution. In its possible failure, it explains the manual installation both under Windows and under Linux. In the last part, the work is devoted to examining the possible improvements of the demo solution and analyzes its security.

Keywords: electronic signature, citizen ID with eletronic chip, web extension

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Elektronizácia administratívnych úkonov má priniesť väčšiu efektivitu, rýchlosť spracovania a vyššiu bezpečnosť vykonávania jednotlivých úkonov. Túto problematiku má každý štát v Európskej únii upravenú podľa seba, preto Európsky parlament vydal nariadenie eIDAS (electronic IDentification, Authentication and trust Services), ktoré má zjednotiť túto legislatívu vo všeobecnej miere. Toto nariadenie je množinou nariadení pre elektronickú identifikáciu a dôveryhodných služieb pre elektronické transakcie na európskom vnútornom trhu, z ktorého vyplývajú povinnosti aj pre našu krajinu, ako napríklad vytvoriť národný identifikačný systém.

V spojitosti na toto nariadenie od Európskeho parlamentu, Slovenská národná rada vydala zákon o e-Govermente. Tento zákon vedie k modernizácii verejnej správy. Jedným zo znakov modernizácie sú občianske preukazy s elektronickým čipom. Tento čip obsahuje identifikačné údaje a údaje na vytvorenie elektronického podpisu, na základe ktorých sa občan dokáže autentifikovať do verejnej správy alebo elektronicky podpísať dokument. Do verejnej správy patria aj univerzity, preto sa v našej bakalárskej práci budeme zaoberať využitím občianskeho preukazu s čipom v univerzitných systémoch.

Navrhneme systém, pomocou ktorého sa bude možné autentifikovať do univerzitného systému alebo ho využiť pre dvojfaktorovú autentifikáciu, čo prinesie vyššiu bezpečnosť a ľahšiu komunikáciu s univerzitnými systémami. Navyše, náš návrh by mal byť schopný vytvárať elektronický podpis, ktorý využijeme či už pri podpisovaní dokumentov alebo emailov.

Analýzou požiadaviek dospejeme k zisteniu, že najlepším návrhom by mohlo byť webové rozšírenie, ktoré by dokázalo interagovať nie len s užívateľom ale aj s webovou stránkou alebo dokonca so serverom.

Na autentifikovanie a vytvorenie elektronického podpisu sa využíva kryptografické funkcie a vyžadujú sa bezpečnostné požiadavky od týchto funkcii. Prvá kapitola bude zaoberať základmi informačnej bezpečnosti. Popíšeme základné bezpečnostné požiadavky a nazrieme do kryptológie a kryptosystémov.

V následnej kapitole popíšeme, aký je rozdiel medzi elektronickým a digitálnym podpisom a pozrieme sa, aké požiadavky kladie zákon na elektronický podpis a ako sa implementuje digitálny podpis pomocou asymetrického kryptosystému.

Tretia kapitola bude zahŕňať popis občianskeho preukazu, čo sa pod tým rozumie

a aké údaje obsahuje. Niektoré občianske preukazy obsahujú elektronický čip, ktorý je nutný pre elektronickú komunikáciu s verejnou správou. Aj tento čip popíšeme v tejto kapitole.

V štvrtej kapitole navrhneme riešenie a popíšeme, na aké problémy môžeme naraziť pri navrhovaní riešenia. Porovnáme aj jednotlivé riešenia a povieme si, prečo niektoré nebudú fungovať alebo sú na implementáciu nevhodné. Pozrieme sa aj na knižnice, ktoré nám umožnia abstrahovať od jednotlivých malých problémov a sústrediť sa iba na vývoj.

Piata kapitola bude obsahovať postup manuálnej inštalácie a odinštalácie lokálnej aplikácie a webového rozšírenia. Opíšeme, ako treba s aplikáciou pracovať, čo umožňuje a ako overiť jej správnu funkčnosť.

V poslednej kapitole sa pozrieme na možné vylepšenia našej demo aplikácie a jej nevýhody. Popíšeme, čo všetko by aplikácia mohla ešte podporovať. Na konci kapitoly by sme sa pokúsili analyzovať bezpečnosť nášho riešenia a bezpečnostné chyby, ktoré by sa mohli dať odstrániť alebo len čiastočne opraviť.

Kapitola 1

Informačná bezpečnosť

V tejto kapitole si definujeme základné bezpečnostné požiadavky a uvedieme základy kryptológie, ktoré budeme potrebovať na pochopenie nášho riešenia.

Terminológia informačnej bezpečnosti ešte nie je ustalená, a preto uvedieme základné pojmy, s ktorými budeme pracovať v celej bakalárskej práci. Pre čitateľov, ktorý majú väčší záujem o túto problematiku, odporúčame knihy [7] alebo [6].

1.1 Bezpečnostné požiadavky

Pri analýze bezpečnostného systému budeme zohľadňovať základné bezpečnostné požiadavky, ktoré sú:

1. **Dôvernosť** - základný bezpečnostný atribút alebo základná požiadavka na ochranu informácie. Zaistenie dôvernosti informácie znamená, že informácia nie je prezradená/odhalená neoprávneným entitám alebo procesom. Na zaistenie dôvernosti sa používajú tak metódy riadenia prístupu k údajom, ako aj šifrovanie.

2. **Integrita** - v úzkom zmysle sa integrita chápe ako ochrana proti neoprávnenej modifikácii alebo zničeniu údajov.

3. **Autentickosť** - vlastnosť vyjadrujúca, že identita entity je tá, ktorá bola deklarovaná.

4. **Nepopretie pôvodu** - bezpečnostná funkcia, pri použití ktorej má príjemca údajov záruku správnosti proklamovanej identity odosielateľa týchto údajov, ktorý vďaka tomu nemôže poprieť, že údaje vytvoril (poslal).

5. **Autentifikácia** - bezpečnostná funkcia, ktorá identifikuje identitu a následne ju aj overí.

6. **Autorizácia** - oprávnenie využívať definované služby chráneného systému, resp. pristupovať definovaným spôsobom k chráneným informáciám.

1.2 Základy kryptológie

Viaceré bezpečnostné funkcie, ktorými sa zapodievame v tejto bakalárskej práci, sú postavené na kryptografických funkciách, preto sa v tejto časti pozrieme na základy kryptológie. Definujeme kryptosystém a typy kryptosystémov, PKI, certifikát a certifikačná autorita.

Kryptológia sa delí na dve časti:

- na **kryptografiu**, ktorá sa zameriava na návrh a skúmanie kryptografických systémov,
- na **kryptoanalýzu**, ktorá sa zameriava na hľadanie slabín v kryptografických systémov a ich využitie.

Kryptografia aj kryptoanalýza sú nutné na navrhnutie dobrého kryptosystému.

1.2.1 Kryptosystémy

Už od minulosti chceli ľudia skrývať informácie pred nepriateľmi alebo len pred svojimi konkurentmi. Tieto dôvody utajovania pretrvávajú do súčasnosti a stimulujú vznik mnohých kryptosystémov. Pod pojmom **kryptosystémom** rozumieme dvojicu kryptografických transformácii (E, D), kde E je šifrovacia transformácia (funkcia, ktorá prevedie vstup do zašifrovanej podoby) a D je dešifrovacia transformácia (funkcia, ktorá dešifruje vstup).[5].

Kryptosystémy je možné rozdeliť podľa viacerých kritérií. V práci ich budeme deliť na symetrické a asymetrické. Pri ich vysvetľovaní využijeme entity Alica a Boba a Evy, ktorá chce do komunikácie medzi Alicou a Bobom zasiahnuť.

Symetrický kryptosystém

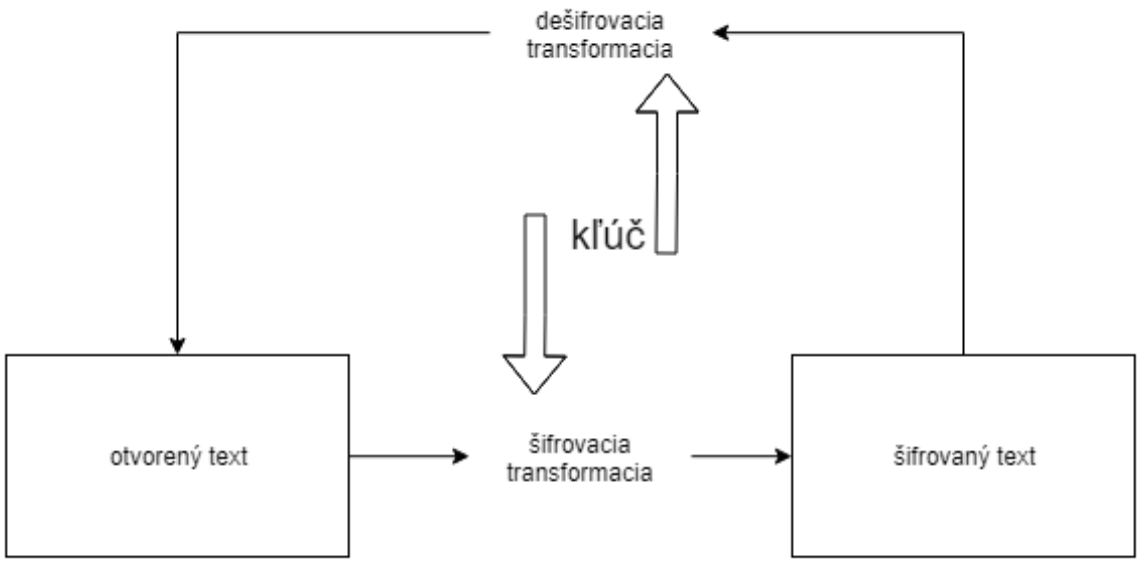
Symetrický kryptosystém je charakterizovaný tým, že šifrovacia a dešifrovacia transformácia zdieľa ten istý kryptografický kľuč, ktorý sa nazýva aj tajný kľuč, pretože ho používajúce entity musia utajiť.

Na obrázku 1.1 je vidieť, že dešifrovacia transformácia je inverzná funkcia k šifrovacej transformácii práve vtedy, keď sa použije rovnaký kľúč v obidvoch transformáciách.

Základnou bezpečnostnou požiadavkou je sila kľúča, ktorá hlavne závisí od jeho dĺžky. Ak dĺžka kľúča je príliš malá, útočník môže prezrieť celý priestor možných kľúčov a týmto spôsobom nájsť správny kľúč na dešifrovanie. Z tohto dôvodu sa v dnešnej dobe používajú dĺžky kľúčov aspoň veľkosti 128 bitov. Pri použití takejto dĺžky[1] je pre utočníka neefektívne[2] použiť útok hrubou silou.

[1] Ak bol kľúč vygenerovaný náhodným spôsobom

[2] Pomer vynaloženého úsilia na získanie informácie a ceny informácie je veľmi malý.



Obr. 1.1: Šifrovanie a dešifrovanie pomocou symetrického kryptosystému

Pozrieme sa, ako prebieha komunikácia medzi Alicou a Bobom, keď Eva chce zistiť informácie, ktoré si posielajú. Alica a Bob chcú chrániť obsah svojej komunikácie pomocou symetrického kryptosystému. Najprv sa musia nejakým spôsobom dohodnúť na kľúči[3], ktorý budú využívať pri šifrovaní a dešifrovaní. Urobia tak pomocou iného kanálu, ktorý nemôže Eva odpočúvať. Dôvernosť komunikácie spočíva na utajení kľúča. Ak Eva správne uhádne alebo zistí kľúč, ktorý používa Alica a Bob na komunikáciu, vie nielen dešifrovať nasledujúcu a predošlú komunikáciu, ale aj vytvárať falošné správy.

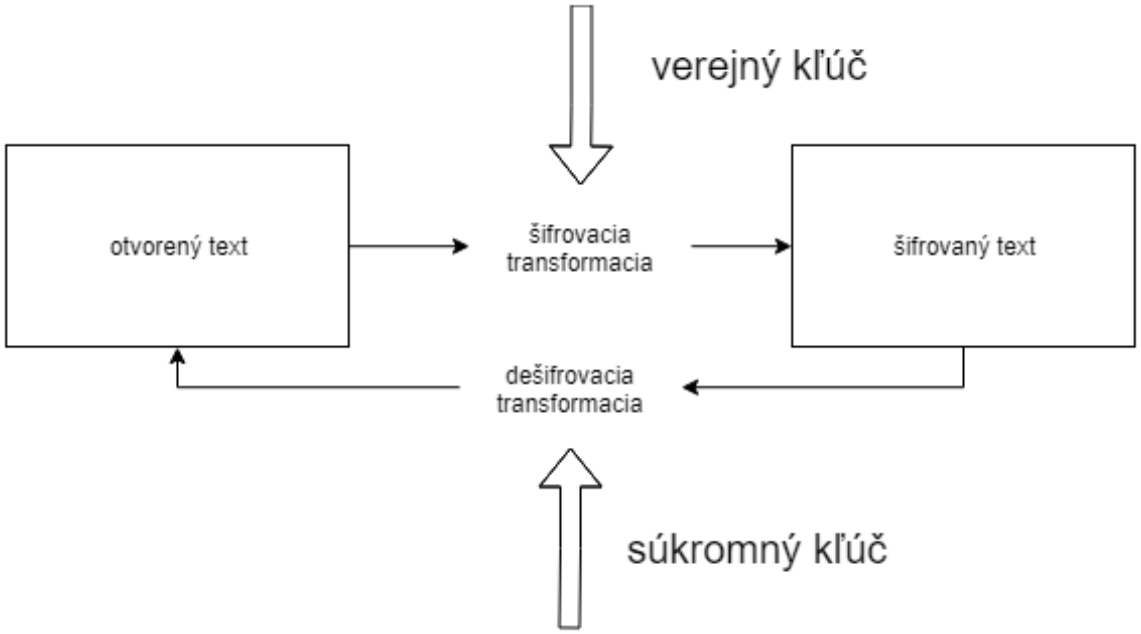
Hlavnou výhodou symetrického kryptosystému je rýchlosť šifrovania dát a nasledovného dešifrovania. Vo viacerých zariadených sú jednotlivé šifry hardwarovo implementované kvôli ich jednoduchosti. Známe sú AES, DES, GOST.

Nevýhodou symetrického kryptosystému je zdieľanie kľúča, ktorý si musia entity nejako bezpečne odovzdať. Distribúcia kľúčov v prípade veľkom počtu používateľov je zložitá a ťažko sa efektívne realizuje. Tento problém rieši práve asymetrická kryptografia.

Asymetrický kryptosystém

Narozdiel od symetrického kryptosystému, ktorý využíva jeden kryptografický (tajný) kľúč, asymetrický kryptosystém používa dva rôzne kryptografické kľúče, z ktorých jeden slúži na šifrovanie a druhý na dešifrovanie dát. Asymetrický kryptosystém je postavený na riešení ťažkých matematických problémov[4], ktorých parametrami sú kryptografické kľúče, ktoré sa z jeden druhého nedajú odvodiť. To umožňuje jeden z kľúčov zverejniť (verejný kľúč) a tento kľúč pozná každý na sieti, na rozdiel od druhého, tzv.

[3]Napríklad ho Bob odovzdá Alici uzatvorenej obálke.
[4]diskrétny logaritmus, faktorizácia prvočísel



Obr. 1.2: Šifrovanie pomocou verejného kľúča a následné dešifrovanie pomocou súkromného kľúča

súkromného kľúča, ktorý si entita uchováva v tajnosti.

Hlavnou výhodou asymetrického kryptosystému je, že Alica a Bob nepotrebujú zabezpečený kanál na výmenu verejných kľúčov. Aby Eva nemohla Bobovi podvrhnúť svoj verejný kľúč a vydávať ho za Alicin, tak na distribúciu verejných kľúčov sa využije riešenie typu PKI (viď 1.3) alebo PGP[5].

Aj keď principiálne máme vyriešený problém distribúcie verejných kľúčov (pomocou PKI alebo PGP), pri implementácii treba vyriešiť aj iné (možnosť útoku so znalosťou otvoreného textu).

Známe asymetrické šifry sú RSA-PKCS1v1.5 alebo ElGamal.

1.2.2 Hashovacia funkcia

Zohráva dôležitú úlohu pri zachovaní integrity a autentickosti dokumentu. **Hashovacia funkcia** je funkcia, ktorá na základe vstupu (bitového vektora) vypočíta tzv. hashovaciu hodnotu (hashword). Argumenty, ktoré spracováva hashovacia funkcia nemávajú spravidla rovnakú dĺžku. Výsledné hashovacie hodnoty sú slová rovnakej dĺžky, spravidla podstatne kratšej ako sú vstupné hodnoty. Vďaka tomu nie je hashovacia funkcia injektívna, t.j. existuje viacero vstupných hodnôt, ktorým je priradená tá istá hashovacia hodnota. Príkladom hashovacej funkcie je zobrazenie, ktoré (textovému) súboru priradí 1 byte, získaný ako suma mod 2 všetkých bytov vstupného súboru. Hashovanie sa využíva pri vytváraní usporiadaných dátových štruktúr a na ochranu integrity

[5] https://sk.wikipedia.org/wiki/Pretty_Good_Privacy

údajov.

V kryptografických konštrukciách sa využívajú tzv. kryptograficky silné hashovacie funkcie, ktoré musia spĺňať nasledujúce požiadavky[4]:

- Sú deterministické, tá istá správa vedie k rovnakému hashu.
- Je možné rýchlo vypočítať hash pre každú danú správu.
- Nie je môžné vygenerovať správu z ich hash hodnoty inak ako skúšaním všetkých možností.
- Malá zmena správy by mala zmeniť každý bit výstupu hashovacích funkcií s pravdepodobnosťou 1/2.
- Je nemožné nájsť dve rôzne správy s rovnakou hodnotou hash.

1.3 PKI (Infraštruktúra verejného kľúča)

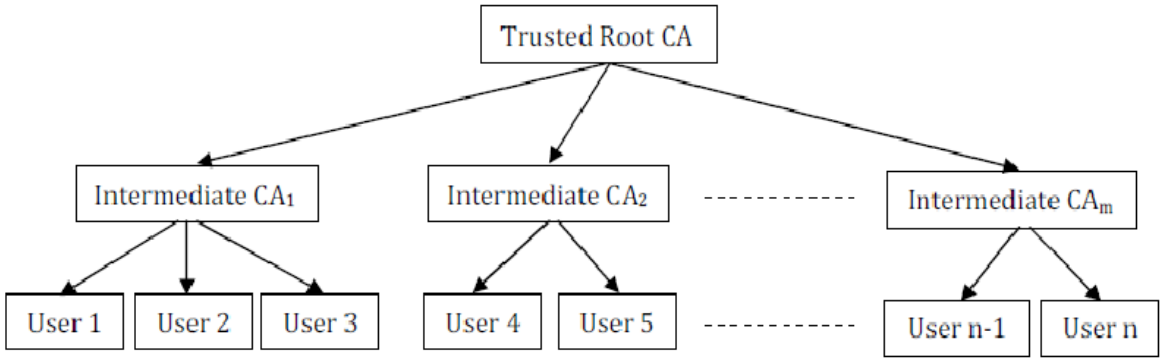
PKI je infraštruktúra verejného kľúča pozostávajúca z certifikačnýách autorít a iných poskytovateľov certifikačných služieb, ktorého úlohou je napomáhať používaniu digitálnych podpisov založených na certifikátoch verejného kľúča (kvôli stručnosti budeme „certifikát verejného kľúča“ označovať pojmom „certifikát“).

Na spojenie identity človeka a verejného kľúča slúži **certifikát**, ktorý je elektronický dokument, ktorým vydavateľ certifikátu potvrdzuje (vo väčšine prípadov certifikačná autorita), že v certifikáte uvedený verejný kľúč patrí osobe, ktorej je certifikát vydaný [2, §6 odsek 1]. Respektíve je to elektronický dokument, ktorý definuje vlastníctvo verejného kľúča právnickej alebo fyzickej osobe. Pre identifikáciu osoby pomocou certifikátu sa nachádzajú v certifikáte identifikačné údaje vydavateľa ako aj držiteľa certifikátu. Certifikát vydáva používateľovi certifikačná autorita.

Certifikačná autorita je poskytovateľ certifikačných služieb, ktorá spravuje certifikáty a vykonáva certifikačnú činnosť [2, §12 odsek 1]. Inak povedané, je to subjekt, ktorý vydáva, spravuje certifikáty a poskytuje certifikačné služby a svojou autoritou potvrdzuje pravosť certifikátov. Na zaistenie autentickosti dát na certifikáte sa využíva digitálny podpis. Na zabezpečenie autentickosti dát na certifikáte slúži digitálny/elektronický podpis (viď kapitola 2) certifikačnej autority.

V našom prípade z asymetrického kryptosystému (viď. asymetrický kryptosystém 1.2.1) by Alica poslala Bobovi svoj certifikát podpísaný certifikačnou autoritou a certifikát certifikačnej autority, ktorým bol Alicin certifikát podpísaný. Ak Eva sa pokúsi nahradiť poslané certifikáty svojimi[6], tak Bob tento podvod rýchlo odhalí, pretože Bob

[6]Eva si vytvorí oba certifikáty a k nim patričné kľúče. Následne jedným súkromným kľúčom podpíše druhý certifikát.



Obr. 1.3: Znázornenie hierarchickej štruktúry PKI, kde na vrchu stromu sa nachádza koreňová certifikačná autorita
(zdroj: https://www.researchgate.net/figure/M-PKI-hierarchical-model_fig4_263526947)

môže mať certifikát certifikačnej autority od iného zdroja, môže ísť na webovú stránku certifikačnej autority, ktorej dôveruje.

Autorít môže existovať viacero a Bob nemusí o všetkých vedieť. Ako má teda Bob dôverovať aj tým certifikačným autoritám, ktoré nepozná? Tento problém rieši **koreňová certifikačná autorita**, ktorá vydáva certifikáty iba certifikačným autoritám. Nasledovne tieto certifikačné autority vydávajú certifikáty ostatným. Takto Bobovi stačí dôverovať iba koreňovým certifikačným autoritám a následne pomocou predávanie dôvery[7] Bob môže dôverovať aj iným certifikačným autoritám, ktorým daná koreňová certifikačná autorita vydala certifikát. Bob takto môže dôverovať aj preňho neznámym certifikačným autoritám.

Pomocou koreňových certifikačných autorít, certifikačných autorít a entity[8] vytvoríme hierarchický model (viď obr. 1.3), kde na vrchu sa nachádza koreňová certifikačná autorita.

Alica má teraz dostupnú metódu ako poslať Bobovi svoj verejný kľúč. PKI Bobovi overiť, že verejný kľúč, ktorý prijal, naozaj patrí Alici. Alica pošle svoj certifikát a celú reťaz certifikátov[9] Bobovi. Bob nasledovne overí všetky certifikáty až ku koreňovej certifikačnej autorite. Až potom Bob môže veriť tomu, že verejný kľúč (nachádzajúci sa v certifikáte, ktorý patrí Alici), ktorý dostal, naozaj patrí Alici. Rovnakým spôsobom bude prebiehať poslanie a overenie Bobového verejného kľúča.

[7]Koreňová certifikačná autorita vydá certifikačnej autorite certifikát. Ak Bob dôveroval koreňovej certifikačnej autorite, tak potom dôveruje aj certifikačnej autorite, ktorej koreňová certifikačná autorita certifikát vydala.

[8]Myslíme osobu alebo inštitúciu, ktorému alebo ktorej bol vydaný certifikát.

[9]Reťaz certifikátov (certificate chain) sú certifikáty od entity až po koreňovú autoritu pospájanú pomocou digitálneho podpisu. Na overenie certifikátu využijeme verejný kľuč z certifikátu o úroveň vyššie.

Zrušenie certifikátov

Certifikačná autorita vydáva certifikáty s určitou dobou platnosti. Platnosť certifikátov môže okamžite zrušiť (pridať do zoznamu zrušených certifikátov). Certifikačná autorita môže zrušiť certifikát z nasledovných dôvodov:

- Stratenie patričného súkromného kľúča danému certifikátu. Entita môže stratiť súkromný kľuč, a preto kvôli bezpečnosti by už nemal daný certifikát platiť (pozn. mohol by ho nájsť niekto neautorizovaný na prácu s tým súkromným kľúčom).
- Ukradnutie kľúča. V tomto prípade môže aj neautorizovaná osoba dešifrovať alebo vytvárať elektronické podpisy, preto sa daný certifikát ihneď pridá do zoznamu zrušených certifikátov.
- Zmena údajov. Entita sa môže rozhodnúť zmeniť svoje identifikačné údaje, čo spôsobí, že certifikačná autorita vydá nový certifikát a zruší platnosť predchádzajúceho certifikátu.
- Z verejného kľúča sa dá hrubou silou nájsť jemu prislúchajúci súkromný kľuč. Tento problém má viacero asymetrických šifier, ale čas, ktorý je potrebný na úspešný útok hrubou silou je obrovský (viac ako 50 000 rokov). Preto existuje obmedzená platnosť certifikátu, ktorá je rádovo menšia (pár rokov).

Podnet na zrušenie certifikátu môže dať úrad, držiteľ certifikátu alebo certifikačná autorita.

Pri overovaní certifikátu musí Alica aj Bob overiť, či prijatému certifikátu neskončila platnosť alebo sa nenachádza v zozname zrušených certifikátov. Tento test platnosti musia spraviť pre každý certifikát z príslušnej reťaze certifikátov vrátane koreňovej certifikačnej autority.

1.4 Zhrnutie

V tejto kapitole sme uviedli základy informačnej bezpečnosti a kryptológie, na ktorých budeme stavať v celej našej bakalárskej práce. Hlavne sa budeme opierať o asymetrické šifrovanie a kryptografické hashovacie funkcie, pomocou ktorých zavedieme digitálny podpis.

Kapitola 2

Podpis v elektronickej forme

Človek už od oddávna dával súhlas svojím slovom, neskôr po vynájdení písma sa súhlas s textom začal vyjadrovať pomocou jedinečného identifikačné znaku nazývaného podpis. Tento prostriedok súhlasu sa používa dodnes. Po rozvinutí elektronického sveta sa ľudstvo snažilo preniesť tento prostriedok súhlasu aj do elektronickej formy a nazýva sa elektronický podpis.

Často sa stáva, že pojmy elektronický podpis a digitálny popis sú nesprávne zamieňané. Elektronický podpis je legislatívny koncept, na rozdiel od digitálneho podpisu, čo je bezpečnostná technológia na vytvorenie elektronického podpisu. Elektronický podpis má byť nezávislý od technológie, ktorá bola použitá na vytvorenie tohto podpisu.

2.1 Elektronický podpis

Ako už vieme, podpis v elektronickej forme sa nazýva elektronický podpis, čo je legislatívny koncept. Aby podpis mal podobné vlastnosti ako vlastnoručný podpis, sú naňho kladené isté bezpečnostné požiadavky zákonom [2, §3 odsek 1.] , ktorý znie:

1. Elektronický podpis je informácia pripojená alebo inak logicky spojená s elektronickým dokumentom, ktorá musí spĺňať tieto požiadavky:
   (a) nemožno ju efektívne vyhotoviť bez znalosti súkromného kľúča a elektronického dokumentu,
   (b) na základe znalosti tejto informácie a verejného kľúča patriaceho k súkromnému kľúču použitému pri jej vyhotovení možno overiť, že elektronický dokument, ku ktorému je pripojená alebo s ním inak logicky spojená, je zhodný s elektronickým dokumentom použitým na jej vyhotovenie,
   (c) obsahuje údaj, ktorý identifikuje podpisovateľa.
2. Podpisovateľ vyhotoví elektronický podpis elektronického dokumentu tak, že na základe svojho súkromného kľúča a elektronického dokumentu vyhotoví nový údaj, ktorý spĺňa podmienky podľa odseku 1.

Stále však nenahrádza vlastnoručný podpis overený notárom, pretože nespĺňa niektoré požiadavky, ako napr. non repudiation of origin (nepopierateľnosť pôvodu), autor môže

tvrdiť, že on ten podpis nevytvoril. Preto vznikol nový typ elektronického podpisu **KEP**.

KEP je skratka pre kvalifikovaný elektronický podpis, ktorý zmenil názov zo **ZEP** (zaručeného elektronického podpisu). Rozdiel medzi **EP** (elektronický podpis) a **KEP** je, že na vytvorenie **KEP** sú kladené vyššie bezpečnostné požiadavky, pomocou ktorých vie **KEP** zabezpečiť aj zvyšné bezpečnostné požiadavky ako autenticita alebo nepopierateľnosť. Vďaka týmto vlastnostiam dokáže nahradiť vlastnoručný podpis overený notárom v elektronickom svete.

2.2 Digitálny podpis

V úvode sme spomenuli, že digitálny podpis je technológia na vytvorenie elektronického podpisu. Táto technológia využíva jednotlivé kryptografické funkcie na dosiahnutie jednotlivých požiadaviek, ktoré sa opisujú už v spomínanom zákone, (viď Elektronický podpis). Na splnenie požiadaviek zákona [2, §3 odsek 1. písm. (a) a písm. (b)] sa využíva konštrukcia založená na asymetrickom šifrovaní a kryptografickej hashovacej funkcii (viď základy kryptológie 1.2). Na splnenie poslednej požiadavky zákona [2, §3 odsek 1. písm. (c)] použijeme certifikát verejného kľúča, ktorý držiteľovi vydala certifikačná autorita.

Teraz sa pozrieme na to, ako pomocou kryptografickej hashovacej funkcii, asymetrického šifrovania a PKI vytvoríme a overíme elektronický podpis. Overenie a vytvorenie sa robí automatizovane pomocou hardvérových zariadení alebo softwarových aplikácií.

2.2.1 Vytvorenie digitálneho podpisu

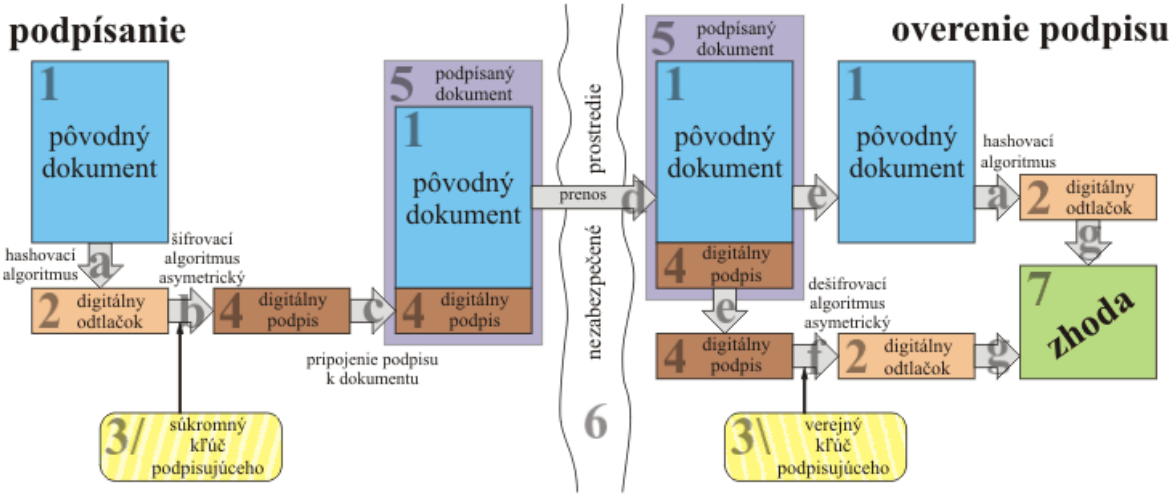
Teraz popíšeme algoritmus, ktorý daná aplikácia používa na vytvorenia digitálneho podpisu z pohľadu Boba.

1. Bob zoberie dokument Doc, ktorý chce podpísať. Následne vytvorí hash H_{Doc} toho dokumentu použitím kryptografickej hashovacej funkcii H.

2. Bob zoberie svoj súkromný kľúč $K_{Bob,privat}$[1] (k tomuto súkromnému kľúču existuje aj verejný Bobov kľúč $K_{Bob,public}$) a pomocou šifrovacej funkcie E vytvorí z hashu digitálny podpis $P_B(Doc) = E(H_{Doc}, K_{Bob,privat})$.

3. Nakoniec Bob priloží podpis $P_B(Doc)$ k dokumentu Doc.

[1] K tomuto súkromnému kľúču musí existovať párový verejný $K_{Bob,public}$, ktorý sa nachádza na certifikáte. Tento certifikát je podpísaný certifikačnou autoritou a deklaruje, že ten verejný kľúč $K_{Bob,public}$ patrí Bobovi.



Obr. 2.1: Na obrázku môžeme vidieť podpísanie elektronického dokumentu a následné overenia daného podpisu
(zdroj: http://epodpis.tuke.sk/digit_podpis.html)

2.2.2 Overenie digitálneho podpisu

Ak nejaká entita potrebuje overiť podpis daného dokumentu. Použije overovací proces. Tento proces si popíšeme z pohľadu Alice.

1. Alica extrahuje verejný kľúč $K_{Bob,public}$ z Bobovho certifikátu.
2. Pomocou verejného kľúča $K_{Bob,public}$ a pomocou dešifrovacej funkcie D Alica dešifruje Bobov podpis P a získa hash H_b.
3. V ďalšom kroku Alica z dokumentu Doc, ktorý Bob podpísal, vypočíta hash H_a.
4. Nakoniec Alica porovná hasha H_a a H_b. Ak sa hashe rovnajú, tak podpis P je správny , inak podpis P nepatrí patričnému dokumentu Doc alebo daný podpis nevytvoril Bob.

2.3 Zhrnutie

V tejto kapitole sme popísali a vysvetlili, čo je to elektronicky a digitálny podpis, aký je rozdiel medzi elektronickým a kvalifikovaným podpisom. Uviedli sme, ako sa vytvára digitálny/ elektronický podpis.

Kapitola 3

Občiansky preukaz

V tejto časti práce popíšeme elektronický preukaz s čipom, uvedieme aké objekty na ňom nachádzajú a ako sa dá použiť v elektronickom svete.

Občiansky preukaz je verejná listina, ktorá slúži na preukázanie totožnosti a štátneho občianstva. Túto verejnú listinu musí mať každý občan, ktorý dovŕšil pätnásty rok veku a má trvalý pobyt na území Slovenskej republiky [3, §2]. To znamená, že každý človek by mohol mať občiansky preukaz s čipom[1], čo by umožňovalo využiť ho na autentifikáciu do univerzitného systému alebo iných systémov, v ktorých sa bude naše riešenie využívať.

Ako už vieme, každý občan nad 15 rokov musí mať občiansky preukaz. Tento občiansky preukaz obsahuje základné údaje o tejto osobe, ako sú meno, priezvisko, rodné priezvisko, pohlavie, štátne občianstvo, dátum a miesto narodenia, rodné číslo a adresa trvalého pobytu občana a elektronický čip [3, §3 odsek 1.].

My sa hlavne zameriame na elektronický čip, ktorý nám umožní preniesť tieto údaje do elektronického sveta.

3.1 Elektronický čip

Pozrieme sa, aké údaje obsahuje elektronický čip na občianskom preukaze. Zákon Slovenskej Republiky umožňuje mať na elektronickom čipe tieto údaje, ktoré možno zapísať do občianskeho preukazu podľa § 3 ods. 1 až 3 a možno zapísať ďalšie údaje v rozsahu a za podmienok ustanovených osobitným predpisom [3, §4]. V našom prípade sa hlavne budeme zaoberať predpisom, zákonom č. 215/2002 o elektronickom podpise, ktorý hovorí o ďalších údajoch, ktoré sa tam môžu zapísať. Tieto údaje môžu byť certifikáty, verejné alebo súkromné kľúče.

Po dlhodobom experimentovaní s elektronickým čipom, sme zistili, že elektronický

[1]Tu môže nastať problém, keď človek môže mať občiansky preukaz s čipom (cudzinci môžu použiť namiesto občianskeho preukazu s čipom eDoPP, o ktorý môže požiadať každý cudzinec s povolením na pobyt na Slovensku), ale nemusí mať vygenerované certifikáty a súkromné kľúče.

Obr. 3.1: Občiansky preukaz s čipom
(zdroj: http://www.b-1.sk/eid-obciansky-preukaz-s-cipom/)

čip obsahuje kryptografické tokeny[2]. Pretože občiansky preukaz obsahuje kryptografické tokeny, tak na prístup k informáciám vyžaduje bezpečnostné požiadavky. Jenda z bezpečnostných požiadavok je zadanie BOK kódu alebo ZEP kódu, ktoré slúžia ako autentifikačný prostriedok. Po zadaní príslušného kódu sme mohli pomocou čítačky a programu pristupovať k jednotlivým údajom použitím Cryptoki API (viď PKCS#11). Občiansky preukaz s čipom obsahuje tieto tokeny:

- Token, ktorý uchováva súkromný kľuč na vytvorenie KEP (viď elektronický podpis 2.1), na ktorého použitie potrebujeme ešte zadať ZEP kód a kvalifikovaný certifikát (odkaz na zákon o kvalifikovanom certifikáte) obsahujúci informácie o držiteľovi občianskeho preukazu a verejný kľuč na overenie KEP podpisu. Aby KEP bol platný, musí byť vyhotovený na bezpečnostnom zariadení podľa zákona [2, §4].

- Na druhom tokene sa nachádzajú dva certifikáty a k nim patričné súkromné kľúče, ktoré sú rozlíšiteľné (tie dva súkromné kľúče) pomocou ID. Z tých dvoch certifikátov, jeden slúži na overovanie elektronického podpisu a jemu patričný kľuč na vytváranie toho podpisu. Druhý certifikát slúži na šifrovanie a jeho patričný kľuč na dešifrovane. Odhadujeme, že tento certifikát sa používa aj so súkromným kľúčom prevažne pri autentifikácii do verejnej správy. Nepodarilo sa nám to, žiaľ, zistiť pre nedostatoku času a pomalej odozvy od štátnych orgánov.

Zaujímavosťou je, že na prístup k certifikátom uložených na týchto tokenoch musíme tiež zadať BOK kód.

Na komunikáciu s kryptografickými tokenmi sa využíva štandard PKCS#11.

PKCS#11 je štandard, ktorý špecifikuje API rozhranie, nazývané **Cryptoki** pre zariadenia, ktoré uchovávajú kryptografické informácie a vykonávajú na nich krypto-

[2]Sú fyzické zariadenia, ktoré sa používajú na uchovávajú kryptografické údaje.

grafické operácie. **Cryptoki** nasleduje[3] jednoduchý objektovo postavený návrh, kde adresuje nezávislosť technológii a zdieľanie zdrojov (viacero aplikácii pristupuje k viacerým zariadeniam) a prezentuje bežný logický pohľad na zariadenia nazývané **kryptografický token** (cryptographic token) [1].

3.2 Zhrnutie

V tejto kapitole sme si popísali občiansky preukaz a elektronický čip. Elektronický čip sa skladá z kryptografických tokenov. Aby sme mohli pristupovať k jednotlivým údajom, sme pred prístupom museli zadať prístupvoý BOK kód.

V budúcnosti sa očakáva, že každý občan nad 15 rokov bude vlastniť občiansky preukaz s elektronickým čipom, čo nám umožní ho využívať na autentifikáciu do elektronických systémov alebo podpisovať elektronické dokumenty.

[3]Pod týmto myslíme, že je narvhnutý týmto spôsobom

Kapitola 4

Návrh a implementácia riešenia

Od začiatku 21. storočia štát začal vydávať občianske preukazy (OP) s kryptografickým čipom, čo ich držiteľom umožňuje vytvárať kvalifikovaný elektronický podpis, elektronický podpis a autentifikovať sa pomocou OP. Ich využitie v informačnom systéme Univerzity Komenského by prinieslo vyššiu bezpečnosť a ľahšiu identifikáciu či autentifikáciu neznámych osôb do nášho systému. Je viacero možností, kde by sa tieto metódy dali využiť, ale my sme sa zamerali iba na tie najdôležitejšie z nich.

- Dvojfaktorová autentifikácia pre administrátorov v systéme Univerzity Komenského.
- Podpisovanie, alebo šifrovanie e-mailov a dokumentov, ktoré sa v informačnom systéme spracovávajú.

Naše riešenie bude zahŕňať obe možnosti a zároveň by malo byť navrhnuté tak, aby ho mohli používať nielen študenti a zamestnanci UK, ale aj tretie strany komunikujúce elektronicky s UK alebo s inou verejnou inštitúciou. Preto sme postavili naše riešenie na všeobecne dostupných internetových prehliadačoch.

4.1 e-Goverment a využitie návrhu

Zo zákona e-Govermente vyplýva viacero povinností pre našu univerzitu ako sú koordinovanie jej databázy s ostatnými databázami orgánov verejnej veci, zriadenie elektronickej schránky, používanie elektronickej podateľne. Ďalej sa zameriava na automatizáciu už existujúcich procesov spracovania informácie, ktoré prebiehajú v tlačovej forme.

Povinnosti vyplývajúce z nariadenia o e-Govermente sú rozsiahla téma bakalársku prácu, preto sme sa rozhodli, že implementujeme iba tú časť, týkajúcu sa využitie elektronického podpisu a autentifikácie v IIKS[1]. (pozn. rozsiahlejšia Analýza Zákona o

[1]Integrovaný informačný a komunikačný systém.

e-Governmente je v bakalárskej práci Zuzany Hromcovej).

Využití nášho návrhu je viacero:

- Dvojfaktorová autentifikácia pre administrátorov.
- Autentifikácia pomocou BOK kódu a občianskeho preukazu s čipom na serveri, povolujúca k dokumentom, ktoré majú byť verejné, ale vyžadujú autentifikáciu.
- Podpísanie dokumentov ako pdf súborov cez webový prehliadač.
- Podpisovanie e-mailov cez webový prehliadač.
- Mnohé iné, ktoré vyžadujú autentifikáciu alebo využitie elektronického podpisu.

Náš návrh sa hlavne zameral na webové prehliadače, pretože to nám umožňuje využiť naše riešenie aj v iných systémoch, nie len v našom IIKS.

4.2 Celkový návrh

Pre správnu komunikáciu medzi užívateľom alebo webovým serverom s eID čipom je nutné zabezpečiť komunikáciu medzi viacerými modulmi a objektmi. To je úlohou systému, ktorý je zobrazený na obr. 4.1. Navrhovaný systém pozostáva z existujúcich a nami vytvorených modulov a objektov, ktorými sme sa zaoberali v tejto bakalárskej práci.

Najdôležitejšou časťou systému je občiansky preukaz, na obrázku 4.1 označený ako ID card, ktorý zabezpečuje bezpečné uloženie súkromných kľúčov a certifikátov verejných kľúčov. Tieto kryptografické kľúče (pozri podkapitolu Základy kryptológie 1.2) slúžia na šifrovanie, dešifrovanie, podpisovanie či overovanie podpisu.

K občianskym preukazom štát vydal aplikáciu, ktorá slúži na autentifikáciu používateľa pomocou občianskeho preukazu s čipom. Táto aplikácia sa využíva na prihlasovanie na portáli www.slovensko.sk[2]. V našej aplikácii využijeme iba eID PKCS knižnicu[3], ktorá poskytuje prístup k smart-card čítačkám.

Jeden z ďalších modulov bude naša lokálna aplikácia (local aplication na obr. 4.1), ktorá sa bude správať ako most medzi webových rozšírením na obrázku s označením extesion a PKCS modulom, akým je na obr. 4.1 eID PKCS Library. Táto aplikácia bude realizovať podpisovanie či iné funkcie, ktoré budú vyžadované od webového rozšírenia na prácu s OP (občianskym preukazom).

Webové rozšírenie prináša užívateľovi možnosti využitia elektronického podpisu. Užívateľ môže použiť tento modul na podpísanie svojho textu, prácu s certifikátmi

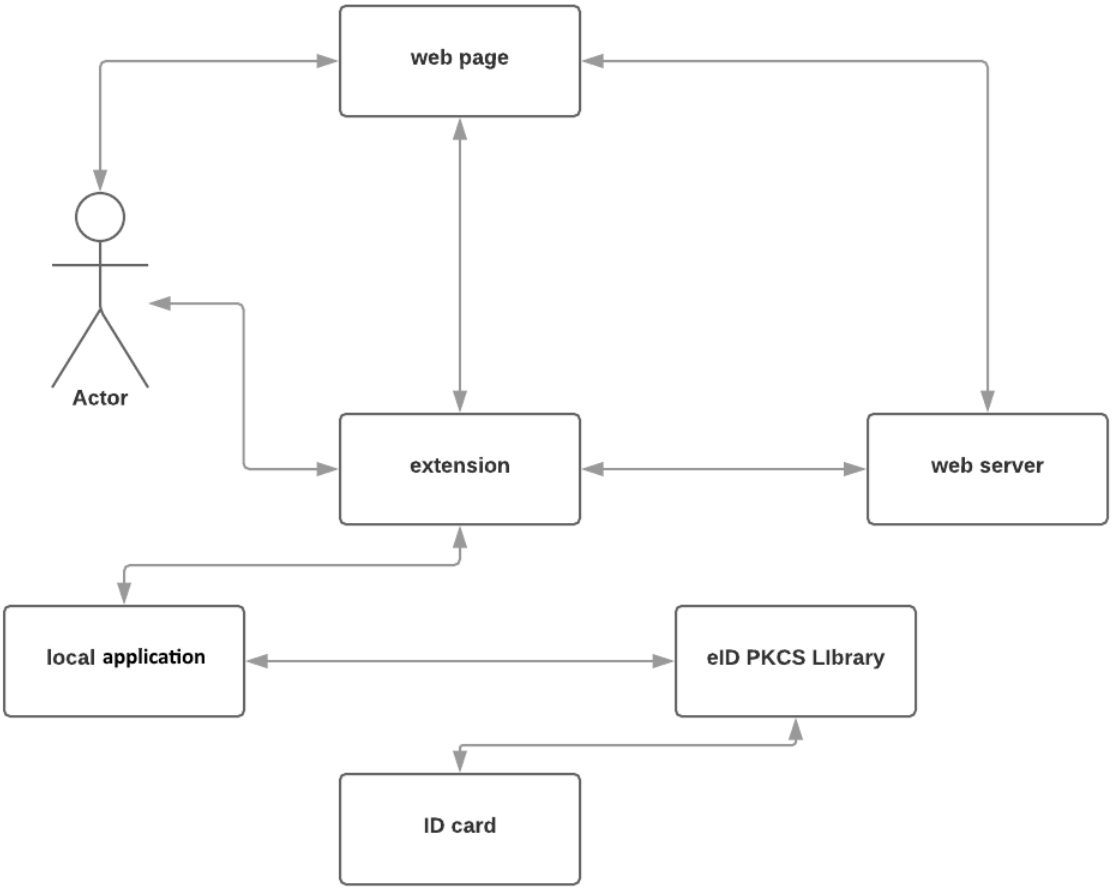
[2](ÚPVS) Ústredný portál verejnej správy

[3]Zdieľaná knižnica od aplikácie eID klienta, ktorá implementuje štandard PKCS #11 (pozri časť PKCS), ktorý zahŕňa funkcie ako vytvorenie, generovanie a mazanie objektov na karte.

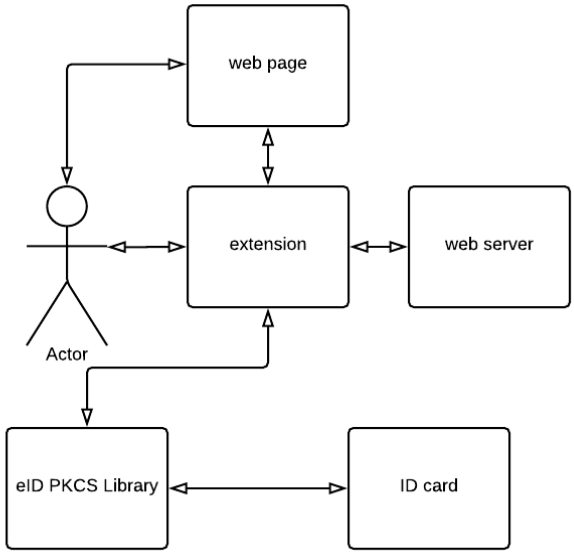
alebo overenie podpisu pomocou certifikátu. Vo väčšine prípadov používateľ bude pracovať s webovou stránkou, ktorá bude nezávisle od používateľa využívať tieto možnosti ako napríklad overovanie podpisu.

Užívateľ bude vo väčšine prípadov pracovať len s webovou stránkou a bude očakávať, že náš navrh bude automatizovať komunikáciu medzi stránkou a webovým rozšírením. Stránka môže samostatne podporovať automatické podpisovanie správ alebo e-mailov. Užívateľ klikne na tlačidlo, ktoré mu odošle podpísaný mail alebo ho autentifikuje pomocou občianskeho preukazu.Aby stránka mohla komunikovať s občianskym preukazom s čipom a využiť jeho objekty, v našom návrhu musí využiť na to webové rozšírenie. Na webovej stránke pravdepodobne bude bežať na pozadí skript, ktorý bude komunikovať s webovým rozšírením pomocou **postMessage**[4] alebo **Xray vision**[5]. Pri návrhu tejto komunikácii treba dať pozor, pretože webové rozšírenie beží ako privilegovaný proces. Kvôli bezpečnosti štandardný návrh izoluje komunikáciu medzi nimi.

[4]https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
[5]`https://developer.mozilla.org/en-US/docs/Mozilla/Tech/Xray_vision`



Obr. 4.1: Obrázok znázorňuje prepojenie jednotlivých modulov a aplikácii v našom návrhu



Obr. 4.2: Obrázok znázorňuje prvý návrh nášho riešenia

Poslednou súčasťou navrhovaného riešenia bude webový server, ktorý môže vyžadovať autentifikáciu pomocou občianskeho preukazu s čipom, podpísanie dokumentu alebo zašifrovanie textu od užívateľa. Na to, aby webový server mohol využívať tieto možnosti, bude pravdepodobne implementovať SAML framework na autentifikáciu užívateľa. Implementovanie podpisovania a overovania dokumnetov sa dorieši na každom serveri samostatne.

4.3 Postup pri návrhu

Pri riešení tohto problému sme prešli viacerými nápadmi, ktoré väčšinou nefungovali alebo boli neefektívne a komplikované na implementáciu.

Z našim prvých nápadov bolo iba webové rozšírenie pre Firefox alebo Chrome. Toto webové rozšírenie by dokázalo komunikovať s webovým serverom a používateľom a zároveň pristupovať k kryptografickým tokenom na občianskom preukaze.

Na začiatku sme sa zamerali iba na Firefox, a na to, aké funkcionality nám ponúka. Do verzie 33, Firefox podporoval modul **window.crypto**[6], ktorý už nie je podporovaný v novších verziách. Nevyzeralo to ako veľký problém, pretože implementácia PKCS podpisovania mohla byť podporovaná cez **signTextJS**[7] rozšírenie. Avšak Firefox sa rozhodol ukončiť podporu **signTextJS**, keď migroval do **WebExtesions**[8], a tým odstránil podporu na **XPCOM-based**[9] webové rozšírenia.

[6] `https://developer.mozilla.org/en-US/docs/Web/API/Window/crypto`
[7] `https://github.com/mozkeeler/signTextJS`
[8] `https://developer.mozilla.org/en-US/Add-ons/WebExtensions`
[9] `https://www-archive.mozilla.org/projects/xpcom/`

Poslednou možnosťou ostávalo využiť pkcs11 api, ktorá je zatiaľ podporovaná iba vo Firefoxe od verzie 58 a nie je úplne dokončená. Toto api podporuje zatiaľ len 4 funkcie:

- **getModuleSlots** – získa meno slotu a ak je prítomný token, tak vráti informácie z toho tokenu
- **installModule** – nainštaluje modul
- **isModuleInstalled** – overí, či je modul nainštalovaný
- **installModule** – odinštaluje modul

Pomocou týchto funkcii dokáže webové rozšírenie načítať moduly do Firefoxu, ktorý potom už sám vie pristupovať k tokenom a využívať ich objekty. Žiaľ, webové rozširenie (extension na obr. 4.1) to nedokáže, pretože pkcs11 api ešte nepodporuje všetky funkcie, ktoré má štandard PKCS #11(viď kapitola PKCS). Po preskúmaní všetkých terajších možností vo Firefoxe sme ani nezačali hľadať iné možnosti v Chrome, pretože on má podobnú filozofiu ako Firefox, tak sme očakávali, že narazíme na rovnaké problémy.

Po zistení, že webový plugin nemôže komunikovať s čítačkou priamo, sme hľadali alternatívne riešenia. Jedným z nich bolo využitie lokálnej aplikácie, ktorá by zastrešovala komunikáciu s PKCS knižnicou (eID PKCS Library, viď obr. 4.1) aj s webovým rozšírením (extension, viď obr. 4.1).

Plugin dokáže komunikovať s lokálnou aplikáciou týmito spôsobmi:

- **native messaging**[10] – webový plugin využíva na komunikáciu **runtime API**, pričom lokálna aplikácia príma vstup a odosiela výstup cez **STDIO**.
- **HTTP Protocol** alebo **Web sockets**[11] – komunikácia prebiehala pomocou siete. Aplikácia vytvorí lokálny sever, ktorý bude bežať na pozadí a plugin bude posielať dotazy cez sieť.

Výhodou native messagingu je, že aplikácia nemusí stále bežať na pozadí, ale iba vtedy, keď plugin potrebuje komunikovať s aplikáciou.

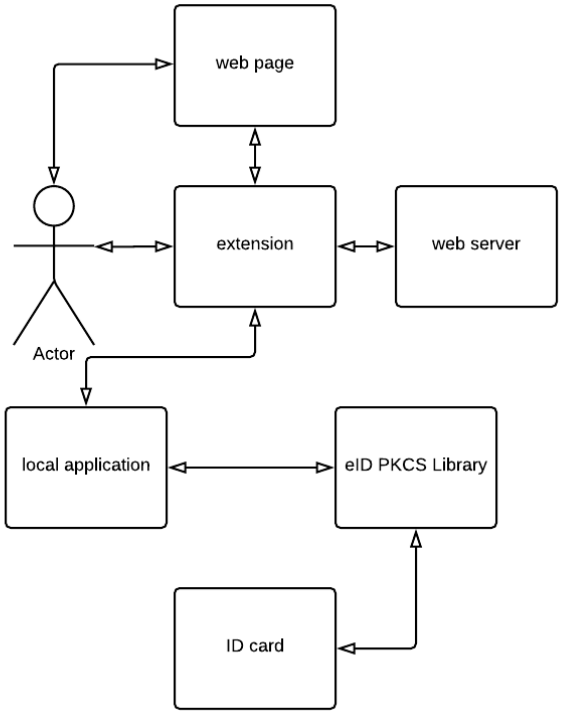
Druhou možnostou bolo využiť native messaging s lokálnou aplikáciou naprogramovanej v pythone. Pri programovaní aplikácie sme mohli využiť na komunikáciu s čítačkou tieto moduly:

- **Python-pkcs11**[12]

[10] `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Native_messaging`

[11] `https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications`

[12] `http://python-pkcs11.readthedocs.io/en/latest/`



Obr. 4.3: Obrázok znázorňuje druhý návrh nášho riešenia aj už s lokálnou aplikáciou

- **PyKCS11**[13]

Python-pkcs11 nemá presne popísané ako získať konkrétny objekt s atribútmi z tokenu. Preto sme sa rozhodli použiť knižnicu PyKCS11, ale ani táto nemala podrobnú dokumentáciu, preto sme museli experimentovať.

Počas testovania lokálnej aplikácie sme narazili na nasledovný problém. Podpis sme nemohli overiť funkciou **verify**, pretože argument funkcie musí byť objekt verejného kľúča a nie certifikát, ktorý sa nachádza na čítačke. Z tohto dôvodu sme vyexportovali certifikát z OP s čipom vo formáte **DER**. Pomocou modulu **Openssl**[14] sme dokázali spracovať certifikát a chceli sme overiť podpis pomocou verejného kľúča z tohto certifikátu.

Overenie podpisu týmto spôsobom nefungovalo. Vzhľadom na to, že nebol dobrý popis ako presne overovanie funguje, tak sme museli niektoré veci overiť samostatne:

- kódovanie textu – skúšali sme len podpísanie byte reťazca. Ako sa neskôr ukázalo, problém nebol v kodovaní textu,
- **Sign** funkcia pridáva nejaký padding alebo má nejakú špeciálnu vlastnosť, ktorá ešte nejako spracováva vstupné parametre funkcie. Rozhodli sme sa podpísať

[13] `https://github.com/LudovicRousseau/PyKCS11`
[14] `https://pyopenssl.org/en/stable/api.html`

rovnaké dáta aj s našou aplikáciu z ročníkového projektu. Hoci vstupné parametre boli rovnaké, podpisy sa líšili.

Zistili sme, že problém neból v kódovaní textu, ale pravdepodobne v podpisovacej funkcii. Táto funkcia má asi skrytú časť, ktorá nie je dobre zdokumentovaná alebo má špeciálne požiadavky na vstupné argumenty.

Kvôli nedostatku času, sme sa rozhodli, že lokálnu aplikáciu naprogramujeme v inom programovacom jazyku, ktorý bude mať dobre zdokumentované api pre PKCS a zároveň bude multiplatformový. Preto sme prešli na jazyk C#.

V tomto návrhu sa nám podarilo obísť problémy s podpisovaním a overovaním podpisu. Bohužiaľ, problém sa našiel pri návrhu komunikáciu cez **native messaging**. Webové rozšírenie môže poslať správu maximálnej veľkosti 4GB a aplikácia môže poslať do 1MB. Tieto hodnoty sa zdali postačujúce, lenže ak už sme chceli poslať správu väčšiu ako 2Kb, tak už to začínalo robiť problémy a spracovanie vstupu na strane C# aplikácie začínalo byť náročné.

Finálne riešenie si berie jednotlivé výhody z predchádzajúcich návrhov. Plugin využíva native messaging iba na spustenie aplikácie, aby nebežala stále na pozadí. Na komunikáciu využíva post požiadavky, ktoré odosiela asynchrónne pomocou knižnice jquery[15].

4.4 Interakcia jednotlivých modulov

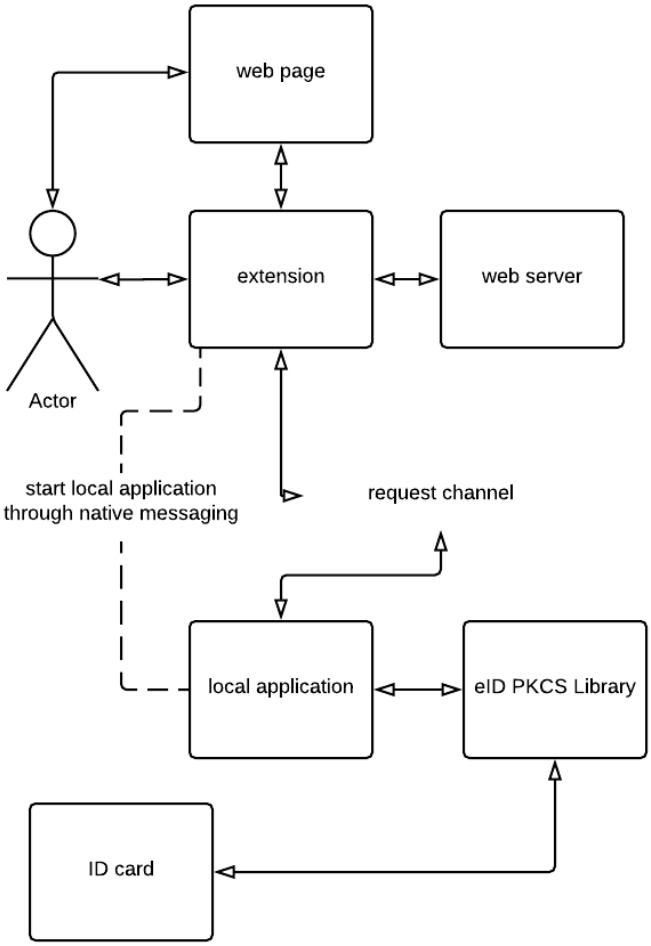
V tejto časti kapitoly si všeobecne popíšeme ako pri špecifických požiadavkách budú jednotlivé moduly komunikovať. Náš systém je navrhnutý tak, že naša lokálna aplikácia poskytuje služby a hociktorý program môže poslať požiadavku, ktorá sa následne spracuje našou lokálnou aplikáciou. Následne naša aplikácia pošle odpoveď na danú požiadavku. Užívateľ komunikuje cez webové rozšírenie viac v kapitole 5.4.

Základou službou je podpísanie textu alebo hashu. Ak užívateľ odošle požiadavku na podpísane textu, webové rozšírenie vygeneruje post požiadavku na adresu `http://localhost:9001/api/sign` s javascriptovým objektom, ktorý obsahuje sha256 hash textu a typ podpisu, ktorý chceme vykonať. Následne webový server lokálnej aplikácie spracuje túto požiadavku nasledovne:

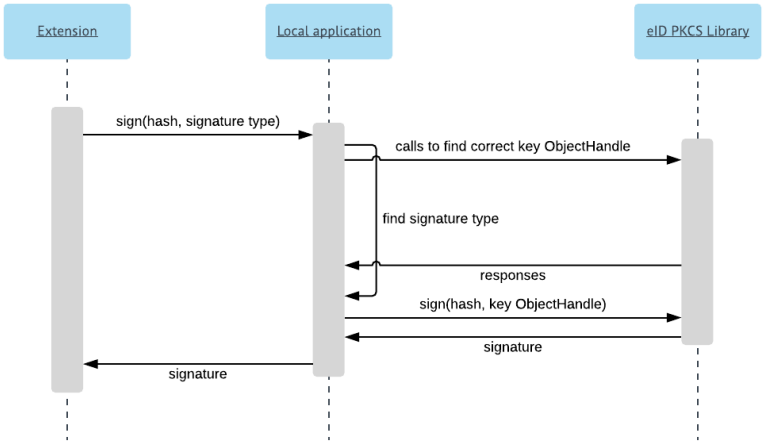
1. Lokálna aplikácia zistí, o aký druh podpisu ide a nájde k nemu správny kľúčový handle.

2. Po získaní kľúču aplikácia zavolá funkciu sign na eID PKCS Library, pretože kľuč nie je extrahovateľný kvôli bezpečnostným dôvodom a získa podpis hashu.

[15] `http://jquery.com`



Obr. 4.4: Obrázok znázorňuje tretí návrh nášho riešenia, kde spustenie lokálnej aplikácie prebieha cez native messaging a posielanie požiadaviek prebieha na sieti



Obr. 4.5: Obrázok znázorňuje interakciu medzi modulmi pri požiadavke sign

3. Následne aplikácia vráti podpis v **BASE64** formáte v **JSON** ako odpoveď na túto požiadavku.

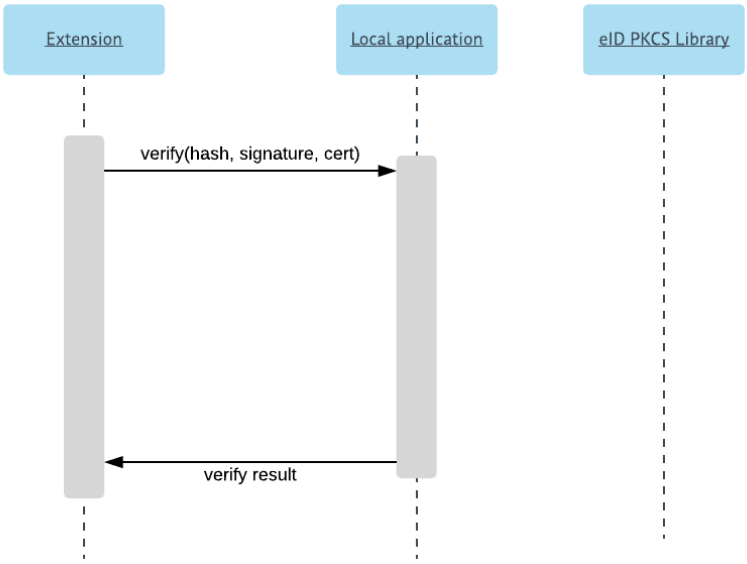
Môže sa stať, že ak aplikácia prvý krát otvára session, bude treba zadať BOK kód cez virtuálnu klávesnicu.

Overovanie podpisu pomocou certifkátu je jedna z ďalších môžností pri ktorej dochádza k interakcii medzi modulmi. Pri tejto službe nie je nutné interagovať **eID PKCS knižnicou**, pretože na overenie podpisu nepotrebujeme žiaden objekt z občianskeho preukazu s čipom. Po odoslaní formulára pre overenie podpisu webové rozširenie odošle post požiadavku na `http://localhost:9001/api/verify` s dátami, ktoré budú obsahovať javascriptový objekt, ktorého atribúty budú:

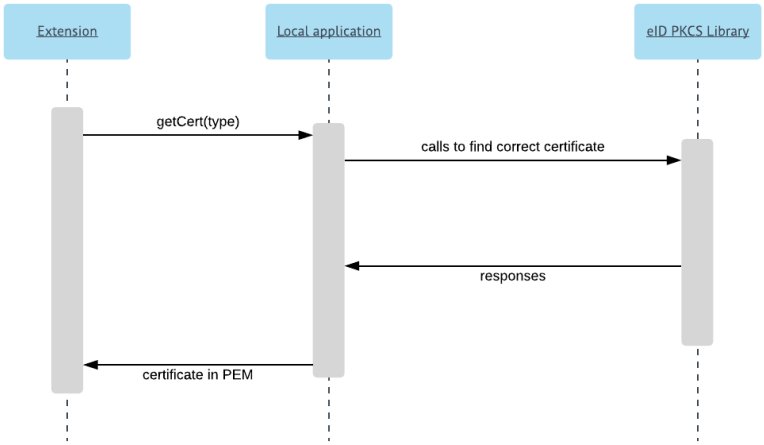
- **Sha256** hash
- Podpis **BASE64** formáte
- Certifikát v **PEM** formáte

Lokálna aplikácia prijme dáta a extrahuje z certifikátu verejný klúč, ktorý použije na overenie podpisu. Ak overovací mechanizmus vrátil true, tak odpoveď na požiadavku bude true, inak sa vráti odpoveď false.

Poslednú možnosť, ktorú naše riešenie ponúka je získanie certifikátu z občianskeho preukazu s čipom, pretože žívateľ po podpísaní textu bude musieť pridať certifikát k danému podpisu, aby niekto iný nemusel hľadať jeho certifikát na overenie jeho podpisu k danému textu. Užívateľ má na výber z troch certifikátov. Vyberie si jeden z nich a odošle post požiadavku na stránku `http://localhost:9001/api/getCert` s dátami, ktoré obsahujú iba typ certifikátu. Nato aplikácia prijme požiadavku a pomocou otvoreného session nájde správny certifikát a vráti ho v **PEM** formáte. Následne ho webové rozšírenie ho vypíše ako text do webového okna.



Obr. 4.6: Obrázok znázorňuje interakciu medzi modulmi pri požiadavke verify



Obr. 4.7: Obrázok znázorňuje interakciu medzi modulmi pri požiadavke getCerts

4.5 Opis riešenia

V tejto časti sa bližšie pozrieme, ako sme naprogramovali webové rozšírenie a lokálnu aplikáciu. Priblížime si ich funkcie a triedy, ktoré popíšeme ich a vysvetlíme,a prečo sme v niektorých prípadoch museli nasledovne postupovať.

Najprv si popíšeme webové rozšírenie a to, z ktorých súborov sa skladá, ktoré časti bežia na pozadí alebo aké časti majú na starosti spracovanie formulárov a nasledovné vytvorenie požiadaviek. V druhej časti popíšeme lokálnu aplikáciu. Popis rozdelíme na statický a dynamický. Pri statickom opise opíšeme jednotlivé triedy a funkcie v súboroch, na rozdiel od dynamického, kde sa pozrieme na interakciu medzi nimi.

4.5.1 Webové rozšírenie

Naše webové rozšírenie je naprogramované iba pre Mozillu Firefox, preto nasledovný popis bude zameraný iba na jeho vývoj v jej prostredí.

Webové rozšírenie môžeme rozdeliť na štyri hlavné časti:

Manifest súbor

Je jediný súbor, ktorý každé webové rozšírenie musí obsahovať. Pomocou tohto súboru špecifikujeme základné metadáta o rozšírení a jeho funkcionalite, ako sú skripty na pozadí, kontent skripty a webové akcie. Súbor je vo formáte JSON s jednou výnimkou, súbor môže obsahovať komentáre v štýle //. Parametre, ktoré nás zaujímajú, sú nasledovné:

1. **background** – obsahuje nasledujúce položky:

   (a) **scripts**– zoznam ciest k javascript súborom, ktoré budú bežať dlhodobo alebo vykonávať operácie nezávisle od života hociktorej stránky,

   (b) **page** – pomocou tohto parametra môžeme nastaviť, že iba niektoré skripty budú bežať na pozadí iba určitých stránok.

   V našom prípade sa v backdround položke bude nachádzať iba jeden súbor background.js (viď časti, ktoré bežia dlhodobo na pozadí).

2. **browser_action** – je to ikona nášho pluginu, na ktorú keď klikneme, sa nám rozbalí menu. Toto menu je normálna webová stránka, na ktorú ukazuje tento kľuč.

   Naše webové rozšírenie obsahuje cestu v tomto kľúči k nášmu webovému menu, ktoré sa zobrazí užívateľovi po kliknutí na ikonu (viď interaktívne časti).

3. **web_accessible_resources** – webové rozšírenie niekedy potrebuje vytvárať nové okná alebo vkladať kód do webových stránok. Na to, aby mohol nájsť cesty k týmto komponentom, musíme to v tomto atribúte zadefinovať.

   Naše webové rozšírenie potrebuje pristupovať k vlastným zdrojom, stránkam, ktoré zobrazí užívateľovi po kliknutí na jednu z možnosti menu, kde window je priečinok obsahujúci samotné webové stránky pre podpísanie, overenie podpisu a získanie certifikátov. Navyše obsahuje skripty, ktoré zabezpečujú komunikáciu s lokálnou aplikáciou a spracovanie formulárov.

4. **permissions** – niekedy aplikácia potrebuje pracovať s webovým stránkami (meniť ich obsah, pristupovať k špeciálnym komponentom). Na takéto správanie aplikácia potrebuje mať povolenia, ktoré vložíme do tejto časti.

   Pre nás je len podstatné povolenie pre **Native Messaging**, ktoré sprostredkúva štartovanie lokálnej aplikácie. Naviac obsahuje aj povolenie pre activeTab, ktoré nevyužíva, ale v budúcnosti sa očakáva vývoj aj týmto smerom. Webové rozšírenie bude meniť obsah aktuálneho tabu.

Časti, ktoré bežia dlhodobo na pozadí

Časti webového rozšírenia, ktoré bežia na pozadí, sú definované v manifest súbore v parametri **Background**. Naše webové rozšírenie má len jeden súbor - *background.js*, ktorý obsahuje iba jeden riadok pre spustenie Native Messaging s lokálnou aplikáciou - pkcsapp. Táto lokálna aplikácia bude bežať celý čas na pozadí, pokiaľ neodinštalujeme rozšírenie alebo nezavrieme Firefoxa spustí sa automaticky štarte Firefoxu.

Popup menu

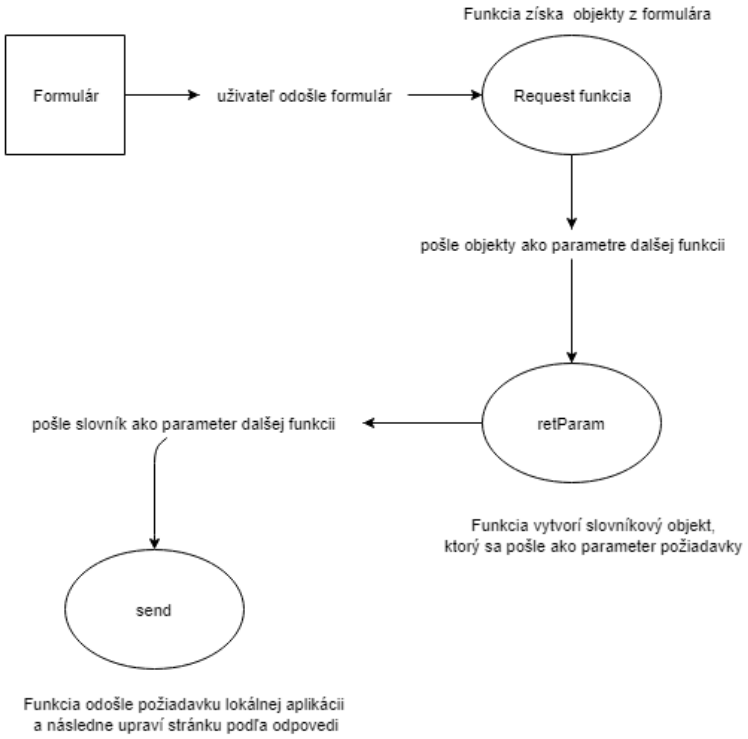
Toto webové menu pozostáva z troch súborov: *choose.html*, *choose.css*, *choose.js*. Prvé dva súbory využívajú šablónu od ukážky **beastify**[16]. Na rozdiel od ukážky súbor **choose.html** načítava aj knižnicu **jquery**[17].

Po načítaní webového rozbaľovacieho menu sa pridajú funkcie k jednotlivým tlačidlám, ktoré čakajú na kliknutie na ich patričné tlačidlo, aby mohli otvoriť nové okno. Všetky sa správajú veľmi podobne, preto si popíšeme iba funkciu sign zo súboru *choose.js*. Po kliknutí na tlačidlo sign sa zavolá funkcia sign (zo súboru *choose.js*). Funkcia najprv získa cestu k window priečinku pomocou funkcie **browser.extension.getURL**[18], ktorý obsahuje jednotlivé webové stránky pre každý typ udalosti.

[16] `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Your_second_WebExtension`

[17] `http://jquery.com`

[18] `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/API/extension/getURL`



Obr. 4.8: Obrázok znázorňuje volania funkcií od odoslania formulára až po prijatia odpovede na požiadavku

Následne otvorí nové stránku v novom okne s url `<cesta,ktorúzískal>/sign/index.html` a uzavrie webové menu.

Interaktívne časti

Všetky interaktívne časti sa nachádzajú iba v priečinku *window*. Každá iteraktívna časť má svoj priečinok (podpriečinok v priečinku window), ktorý obsahuje potrebné súbory na obslúženie danej interakcie[19]. Na vyvolanie danej interakcie existuje tlačidlo z menu (po kliknutí na dané tlačidlo sa spustí daná interakcia). Priečinok *window* obsahuje aj súbor *config.js*. Tento súbor obsahuje zdieľané premenné a funkcie. Jedna z nich je asynchrónna funkcia **sha256**[20], ktorá zoberie text a vráti jeho hash v hex formáte.

Popíšeme si len skriptové časti, pretože tie vykonávajú činnosti, ktoré tvoria jadro našej bakalárskej práce. Každá stránka má svoj vlastný skript, ktorý čaká na odoslanie formulára (viď obrázok 4.8). Po odoslaní každý skript získa jednotlivé časti formulára, čo má na starosti funkcia **request**, a odovzdá ich asynchrónnej funkcie **retParam**, ktorá spracuje tieto parametre do objektu s atribútmi, ktorý sa pošle požiadavku lo-

[19]Ako sú HTML, JS alebo CSS súbory, ktoré spolu vytvárajú webovú stránku použitím knižnice Boostrap (https://getbootstrap.com)

[20]`https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/digest`

kálnej aplikácii na spracovanie, pomocou funkcie **send**. Následne funkcia **send** počká na odpoveď a podľa nej upraví webstránku.

Funkcie retParam a send sme zvolili asynchrónne, pretože jednotlivé úkony môžu trvať aj viac sekúnd. Napríklad čakanie na odpoveď post požiadavky môže zdržať zadávanie BOK kódu užívateľom.

4.5.2 Lokálna aplikácia

Najdôležitejšou časťou nášho riešenia je lokálna aplikácia, ktorá sprostredkúva komunikáciu medzi občianskym preukazom s čipom a webovým rozšírením. Pre tento program sme zvolili programovací jazyk C#, kvôli jeho podpore rôznych platforiem a nuget packages[21], ktoré umožňujú ľahký manažment závislostí a obsahujú predprogramované knižnice. Tieto knižnice nám umožnili abstrahovať od implementačných detailov a umožnili nám navrhnúť aplikáciu, ktorá sa dá ľahko upraviť alebo sa dajú do nej rýchlo doprogramovať ďalšie funkcie.

Pre pochopenie funkčnosti našej aplikácie sme rozdelili jej popis na statický a dynamický.

- V statickej časti popisu sa pozrieme na jednotlivé triedy, knižnice a návrhové vzory, ktoré sme použili.
- V dynamickej časti sa pozrieme, ako komunikujú časti aplikácie medzi sebou a zdôvodníme si, prečo sme zvolili niektoré návrhové vzory.

Statický popis

Túto časť si rozdelíme na dve podčasti, kde v jednej sa pozrieme na komunikáciu s OP s čipom a hľadanie objektov a párovanie certifikátov s ich súkromnými kľúčmi alebo samostatné podpisovanie a overovanie. V druhej časti si priblížime implementáciu serverovej časti a využitie knižnice **Self-Host**, ktorá nám umožnila abstrahovať od sieťovej komunikácie a iba sa zamerať na spracovanie post požiadaviek na podpisovanie, získanie certifikátu a iné požiadavky, ktoré môžu vyžadovať interakciu s OP.

Komunikácia s OP V tejto časti sa zameriame na moduly a triedy, ktoré komunikujú alebo inak prispievajú ku komunikácii s občianskym preukazom s čipom. Táto časť používa tieto dva moduly:

- **Ini-parser**[22]– zabezpečuje nám ľahké extrahovanie kľúčov a sekcií z iných súborov. V našom prípade sa jedná iba o extrahovanie cesty k eID klient modulu zo súboru config.ini

[21] `https://www.nuget.org`
[22] `https://github.com/rickyah/ini-parser`

- **Pkcs11Interop**[23] – knižnica, ktorá implementuje **PKCS#11 API** (viď. sekcia PKCS) do **.NET**. Umožní načítať **PKCS#11** modul a volať funkcie z neho,

Teraz si popíšeme súbory, z ktorých sa táto časť skladá.

1. *Pkcs11Manager.cs*

   Jadro celej komunikácie s čítačkou prebieha v tomto súbore. Tento súbor obsahuje singleton triedu Pkcs11Manager. V jej konštruktore využijeme knižnicu **Iniparser** a vyextrahujeme cestu k modulu eID klienta a vytvorí objekt **Pkcs11** (viď knižnica Pkcs11Interop). Dôležité je podotknúť, že pri vytváraní tohto objektu musíme povedať konštruktoru, aby objekt podporoval viac vláknové prístupy kvôli serverovej časti. Ďalším problémom bolo prihlásenie sa pomocou funkcie **Login**. V prvom riešení sme použili ako druhý argument funkcie prázdny reťazec. Toto riešenie fungovalo iba pod OS Windows. Modul otvoril okno **VirtualKeyboard**, kde sme mohli zadať heslo. Pod Linuxom to hlásovalo chybu nesprávny **PIN**. Po prečítaní štandardu, kde je napísané: „Or the user might not even use a PIN authentication could be achieved by some fingerprint-reading device, for example. To log into a token with a protected authentication path, the pPin parameter to C_Login should be NULL_PTR.“(`https://www.cryptsoft.com/pkcs11doc/v220/group__SEC__11__6__SESSION__MANAGEMENT__FUNCTIONS.html`), sme zmenili druhý argument z prázdneho reťazca na **null**. Po tejto zmene to začalo fungovať aj pod Linuxom.

   Trieda obsahuje tieto funkcie:

   - **SignEp**

     Jednou z hlavných funkcii je funkcia signEp. Táto funkcia nájde slot, ktorý obsahuje správni **key_ID** a s ním podpíše dáta.

     Po prihlásení sa do tokenu, funkcia vyhľadá náš súkromný kľuč pomocou id **key_ID**. Ak taký nenašla, tak to znamená, že je v zlom tokene a musí prezrieť iný. Podpisový mechanizmus sme zvolili **CKM_RSA_PKCS**[24], pretože naše občianske preukazy podporujú iba tento.

   - **getCertificates**

     Na získanie všetkých certifikátov z občianskeho preukazu sme naprogramovali túto funkciu. Táto funkcia prehľadá všetky tokeny na karte a vráti všetky certifikáty vo formáte slovníka, kde kľuč bude meno tokenu, z ktorého ho získala a hodnota bude zoznam certifikátov, ktoré sa nachádzali na

[23] `https://pkcs11interop.net`

[24] `https://www.cryptsoft.com/pkcs11doc/v100/group__SEC__13__MECHANISMS.html#CKM_RSA_PKCS`

danom tokene. Certifikáty budú už zaobalené v našej triede **Certificate** pre jednoduchšiu manipuláciu s nimi.

- **getTypeCertificate**

  Vráti certifikát na overenie podpisu z daného tokenu.

2. *Certificate.cs*

   Tento súbor obsahuje triedu **Certificate**, ktorá obaľuje triedu X509Certificate2[25] a pridáva nové funkcie ako **verifyHash**, **verifyData**, **getPEM**, ktoré nám uľahčujú prácu s ním.

   - Funkcia **verifyHash** resp. **verifyData** overí hash z text resp. dáta pomocou verejného kľúča extrahovaného z certifikátu.
   - Pomocou funkcie **getPEM** získame PEM formát certifikátu.

3. *Constatns.cs*

   Súbor, ktorý obsahuje všetky dôležité nastavenia konštanty. Navyše obsahuje aj globálne funkcie ako **HexStringToByte** a **RemoveBeginEndCert**, ktoré sa využívajú globálne a sú statické.

Serverová časť Hlavnú časť riešenia tvorí knižnica OWIN[26]. Táto knižnica nám umožnila sa sústrediť iba na spracovanie požiadaviek a neriešiť serverovú stránku aplikácie. Najprv sme ju museli správne nakonfigurovať. Na to slúži súbor *StartUp.cs*. Väčšina kódu v tomto súbore je z tutoriálu[27]. Navyše, aby naše webové rozšírenie sa mohlo pripojiť na našu lokálnu aplikáciu, sme museli nastaviť **CORS**[28] pomocou funkcie **EnableCORS**[29] a ako parameter bol objekt **EnableCorsAttribute**.

Spracovanie požiadaviek prebieha v súboroch, ktoré sa nachádzajú v priečinku Commands. Súbory obsahujúce v mene Controller majú triedy na spracovanie požiadaviek. Typ požiadavky, ktorá sa spracuje ,je definovaná ako text v mene súboru pred retazcom Controller. Napríklad, súbor SignController bude spracovávať požiadavku `http://localhost:9001/api/sign`, pretože pred **Controller** textom sa nachádza reťazec Sign. Keby sme chceli zmeniť text **Controller** na niečo iné, nastavíme to v súbore *StartUp.cs* v parametri **routeTemplate**, kde v zátvorke namiesto Controller vložíme iný text. Po tomto kroku by sme ale museli premenovať všetky triedy a súbory, ktoré obsahujú v mene Controller na náš nový text, ktorý sme tam vložili.

[25] `https://msdn.microsoft.com/sk-sk/library/system.security.cryptography.x509certificates.x509certificate2(v=vs.110).aspx`

[26] `http://owin.org`

[27] `https://docs.microsoft.com/en-us/aspnet/web-api/overview/hosting-aspnet-web-api/use-owin-to-self-host-web-api`

[28] `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`

[29] `https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api`

V priečinku *Commands* sa nachádzajú aj súbory obsahujúce text **Command**. Tieto súbory obsahujú triedy, ktoré slúžia ako automatický balič dát z post požiadavkou pre **OWIN**. Nám to umožňuje neriešiť spracovanie **JSON** textu do objektov.

Teraz si popíšeme, ako vo všeobecnosti vyzerá trieda na spracovanie požiadavky. Každá z nich musí mať v mene triedy text Controller, musí byť public a musí dediť od triedy **ApiController**. Každá trieda môže obsahovať jednotlivé funkcie na spracovanie rôznych požiadaviek z rozdielnym počtom argumentov. Funkcie môžu obsahovať aj ako vstupný argument objekt, ale jeho atribúty musia byť public a každá z nich musí mať funkcie set a get. Navyše meno premennej sa musí zhodovať s menom atribútu v **JSON** formáte, ktorý server bude prijimať. Návratová hodnota funkcie môže byť tiež objekt.

Dynamický popis

V statickom popise sme si uviedli, ako sú naprogramované jednotlivé funkcie a aké knižnice sme použili. V tejto časti sa pozrieme ako medzi sebou sa tieto triedy volajú.

Aplikácia prijme post požiadavku na adresu `http://localhost:9001/api/` s javaskriptovým objektom, ktorý obsahuje parametre pre konkrétne volanie. Následne **OWIN** server konvertuje vstup do konkrétnej **Command** triedy, ktorá slúži ako argument danej post funkcie.

Naša lokálna aplikácia podporuje len tri požiadavky:

1. Požiadavka pre podpis

   do SignCommand objektu **OWIN** vloží všetky argumenty volania, a nato sa zavolá funkcia post zo **SignController**. Táto funkcia si vytvorí inštanciu **Pkcs11Manager**. Kvôli tomu, že Pkcs11Manager pristupuje k čítačke a vykonáva operácie s nimi, tak sme ho navrhli ako singleton. Následne po získaní inštancie Pkcs11Manager funkcia zistí o aký podpis ide a podľa toho to podpíše.

   Ak užívateľ zvolil podpis **SIG_EP**, tak aplikácia musí nájsť správny súkromný kľúč na podpísanie. Tu vzniká problém. Dva súkromné kľúče sa nachádzajú na rovnakom tokene a majú rovnaké atribúty, pričom jeden z nich je náš hľadaný. Na správne nájdenie tohto kľúča sme najprv našli jeho príslušný certifikát, ktorý sa odlišoval od toho druhého certifikátu v tom, že mohol byť použitý na overovanie podpisu. Pomocou tohto certifikátu a jeho id sme mohli nájsť jeho prislúchajúci kľúč a následne podpísať hash. Pri typu podpisu SIG_ZEP nevznikol tento problém, pretože súkromný kľúč s certifikátom má vlastný token na karte. (pozn. Funkcia SIG_ZEP nebola implementovaná, pretože je úplne identická s sign s rozdielom druhého prihlásenia, čo vyžaduje karta pri využití tohto kľúča na podpísanie.)

2. Požiadavka na overenie podpisu

Pri spracovaní tejto požiadavky vôbec nepotrebujeme komunikáciu s OP, preto ani nezískavame inštanciu z triedy PKCS11Manager. Funkcia pre spracovanie post požiadavky načíta certifikát z PEM reťazca do Certificate Objektu. Tento objekt obsahuje funkciu verify, ktorá overí daný podpis, a to vráti ako odpoveď na požiadavku.

3. Požiadavka na získane certifikátov

   Lokálna aplikácia prijme post požiadavku na url adresu `http://localhost:9001/api/getcerts` s objektom obsahujúci typ certifikátu, ktorý chceme získať z občianskeho preukazu. Ako v predchádzajúcich prípadoch sa o prerobenie vstupu do objektu sa postará OWIN knižnica. V tomto prípade funkcia post v **GetCertController** prijme objekt typu **GetCertCommand**, ktorý obsahuje len reťazec typu certifikátu. Následne funkcia Post získa inštanciu triedy **Pkcs11Manager**, na ktorej následne zavolá funkciu **getCertificates**, ktorá mu vráti všetky certifikáty na občianskom preukaze. Funkcia **getCertificates** prejde všetky tokeny a zo všetkých extrahuje certifikáty a uloží ich do triedy Certificate pre lepšiu manipuláciu s nimi. Nakoniec podľa typu požiadavky vráti daný certifikát v **PEM** formáte.

Záver

V tejto kapitole sme sa pozreli na spojitosť nášho riešenia s e-govermnetom a jeho využitím v IIKS. Následne sme popísali jeho návrh a postup, ako sme sa k nemu dopracovali. Implementovali sme demo riešenie, ktorá je ľahko rozšíriteľná a ukazuje, že naše riešenie je funkčné a prakticky použiteľné.

Kapitola 5

Inštalácia a práca s riešením

V tejto kapitole sa pozrieme, ako manuálne alebo automaticky nainštalovať aplikáciu. Ďalej uvedieme ako pracovať s riešením a otestovať správnu funkčnosť nášho riešenia.

5.1 Inštalácia lokálnej aplikácie

Pred inštáláciou našej lokálnej aplikácie musíme najprv overiť, či máme nainštalované potrebné moduly alebo iné aplikácie.

5.1.1 Požiadavky

Pred inštaláciou a spustením aplikácie skontrolujeme, či máme k dispozícii:

- čítačku kariet a nainštalovaný ovládač,
- nainštalovanú aplikáciu eID klient,
- Mozillu Firefox,
- občiansky preukaz vybavený čipom a s nastaveným ZEP a BOK kódom.

5.1.2 Podporované operačné systémy

Aplikácia je zameraná iba na Linux a OS Windows operačné systémy. Kvôli nedostatku času bola odskúšaná na Windows 10 a Ubuntu 17.04. Odhadujeme, že na rôznych verziách by tiež mohla fungovať bez problémov alebo len s malou úpravou.

5.1.3 Automatická inštalácia

Na zjednodušenie inštalácie lokálnej aplikácie sme vytvorili automatický inštalátor vo **WIX toolset**[1] pre OS Windows a na Linux sme použili **DEB**[2] balíčky. Tieto automatické inštalátory sú len na ukážku a neriešia všetky prípady, preto na niektorých operačných systémoch bude treba využiť manuálnu inštaláciu (viď sekcia manuálna inštalácia.)

Windows

Pre správnu inštaláciu aplikácie pod OS Windows je nutné overiť, či máme nainštalované .NET framwork. Potom stačí spustiť aplikáciu *PkcsApp.Installer.msi*, ktorá nainštaluje aplikáciu do Program Files a nastaví registre, aby fungoval native messaging. Ak pri inštalovaní aplikácie vzniknú chyby, prejdeme na manuálnu inštaláciu.

Linux

Pred tým než začneme inštalovať aplikáciu pod Linuxom, treba overiť, či máme nainštalované Mono v opačnom prípade ho nainštalujeme. Nakoniec pomocou príkazu **sudo dpkg -i pkcsapp.deb** aplikáciu nainštalujeme.

5.1.4 Manuálna inštalácia

Automatická inštalácia uľahčuje inštalovanie, no nerieši všetky prípadné problémy. Ak sa dá, uprednostníme automatickú inštaláciu kvôli jej jednoduchosti a rýchlosti. Teraz si popíšeme kroky pre manuálnu inštaláciu pod OS Windows 10.

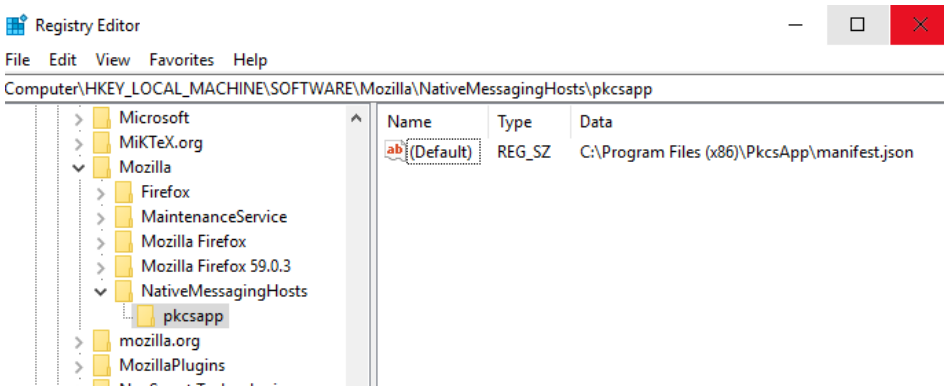
Windows

Rozbalíme súbor *pkcsapp.zip* do priečinku Program Files. Po rozbalení sa nám vytvorí nová zložka PkcsApp. Po jej otvorení by sme mali vidieť celú aplikáciu a jej pomocné súbory. Vyhľadáme v nej *config.ini*, v ktorom zmeníme hodnotu atribútu path na cestu k modulu od eidklient. Nájdeme absolútnu cestu k eidklient priečinku. Napríklad ak máme nainštalovaný eidklient v Program Files v zložke eID klient, tak absolútna cesta k nemu bude `C:\ProgramFiles\eIDklient`.

Atribút path v *config.ini* bude vyzerať nasledovne `<cestakeIdklientu>\pkcs11_x86.dll`. V ukážkovom prípade bude vyzerať takto: `C:\ProgramFiles\eIDklientpkcs11_x86.dll`

Poslednou vecou je nastaviť v registroch **Native Messaging**, aby webové rozšírenie mohlo samovoľne spúšťať aplikáciu.

[1] `http://wixtoolset.org`
[2] `https://wiki.debian.org/HowToPackageForDebian`



Obr. 5.1: Nastavenie native messaging v registroch pod OS Windows

1. Vyhľadáme a spustíme aplikáciu *regedit.exe*.
2. Vytvoríme nový key, `HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\NativeMessagingHosts\pkcsapp` (viď obr. 5.1).
3. Kľuč alebo key by mal mať iba jednu default hodnotu (viď obr. 5.1). Túto hodnotu nastavíme na cestu k súboru manifest.json, ktorý sa nachádza na mieste, kde sme rozbalili aplikáciu.

Teraz máme nainštalovanú aplikáciu a prejdeme na inštalovanie webového rozšírenia do Firefoxu (viď inštalácia webového rozšírenia).

Linux

Rozbalíme súbor *pkcsapp.zip* do priečinku `/usr/bin/`. Rovnako ako OS Windows musíme prenastaviť hodnotu atribútu path v súbore *config.ini* na absolútnu cestu k modulu od eID klientu. Ak máme eID klient nainštalovaný v súbore `/usr/lib/eidklient/`, tak atribút nastavíme na `/usr/lib/eidklient/libpkcs11_sig_x64.so`.

Vo všeobecnosti by sme nastavili atribút takto `<cestakeIdklientu>/libpkcs11_sig_x64.so`.

Zostáva nám už len nastaviť native messaging. Vystrihneme súbor **pkcsapp.json**, ktorý sa nachádza v zložke, kde sme nainštalovali našu aplikáciu a vložíme ho do súboru `/usr/lib/mozilla/native-messaging-hosts/`. Môže sa stať, že priečinok *native-messaging-hosts* nemusí existovať, v tomto prípade ho treba vytvoriť. Otvoríme súbor *pkcsapp.json* a zmeníme v ňom hodnotu atribútu path tak, aby obsahovala absolútnu ceste k súboru *PkcsApp.exe*. Hodnota atribútu path bude vyzerať takto: `<cestakeIdklientu>/PkcsApp.exe`.

Týmto spôsobom by mala byť aplikácia správne nastavená a nakonfigurovaná. (pozn. Odkaz ako nastavovať native messaging nájdeme na stránke `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Native_manifests`)

5.2 Inštalácia webového rozšírenia

Webové rozšírenie je možné nainštalovať 2 spôsobmi. Popíšeme si najprv ten jednoduchší z nich.

1. Spustíme Firefox a otvoríme stránku `http://addons.mozilla.org/`.
2. Dáme vyhľadať **PkcsApp** a nainštalujeme.

Druhý spôsob je trochu komplikovanejší. Tento spôsob načíta rozšírenie iba dočasne, kým nezavrieme Firefox.

1. Skopírujeme *extesion.zip* do počítača.
2. Spustíme Firefox a napíšeme do hlavného okna **about:debugging**.
3. Klikneme na tlačidlo **Načítať dočasný doplnok**.
4. Vyhľadáme *extesion.zip*, ktorý sme nakopírovali do počítača a otvoríme.

5.3 Odinštalácia lokálnej aplikácie

Ak sme nainštalovali aplikáciu cez automatickú inštaláciu, tak jej odinštalácia bude tiež jednoduchá.

5.3.1 Automatická odinštalácia

Windows

Na odinštalovanie aplikácie postupujeme nasledovne:

1. Otvoríme **ovládací panel** a prejdeme do časti **Pridať alebo odstrániť program**.
2. Nájdeme **PkcsApp** program a odinštalujeme.

Linux

Na odinštalovanie aplikácie na Linuxe, postupujte nasledovne:

1. Spustite príkazový riadok.
2. Do príkazového riadku napíšte **sudo dpkg -r pkcsapp**.

5.3.2 Manuálna odinštalácia

Ak sme použili manuálnu inštaláciu, tak na odinštalovanie aplikácie musíme zvoliť manuálnu odinštaláciu.

Windows

Pri manuálnej odinštalácii postupujeme takto:

1. Nájdeme priečinok, kde sme nainštalovali **PkcsApp** aplikáciu a nasledovne ho odstránime.
2. Vyhľadáme a spustíme aplikáciu *regedit.exe*.
3. 3. Odstránime kľuč `HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\NativeMessagingHosts\pkcsapp` (viď obr. 5.1).

Linux

Linuxová manuálna odinštalácia prebieha nasledovne:

1. Vymažeme priečinok **pkcsapp**.
2. Nakoniec vymažeme súbor `/usr/lib/mozilla/native-messaging-hosts/pkcsapp.json`.

5.4 Práca s návrhom

Táto bakalárska práca je zameraná na riešenie problému komunikácie webovým prehliadačom a eID kartou. Je to ukážka, ktorá znázorňuje, ako v budúcnosti navrhovať a implantovať nové aplikácie, ktoré budú vyžadovať komunikáciu s eID kartou.

5.4.1 Štartovanie a práca s aplikáciou

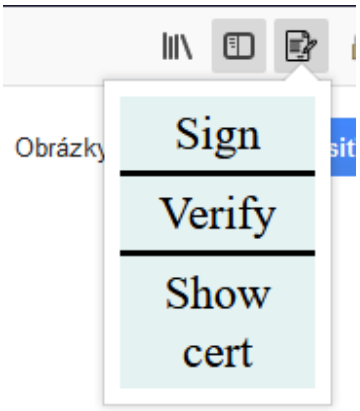
Na začiatku pripojíme elektronickú čítačku s občianskym preukazom k počítaču. Následne spustíme Firefox, kde klikneme na ikonku aplikácie v pravom hornom rohu (viď. obr. 5.2).

Rozbalí sa nám kontextové menu. Na tomto menu máme na výber z nasledujúcich troch možností:

1. **Sign** - podpísať

   Kliknutím na tlačidlo **Sign** z menu sa nám otvorí nové webové okno (viď. obr. 5.3).

   V tomto novom okne sa bude nachádzať formulár, ktorý sa zakladá z 3 častí:



Obr. 5.2: Menu webového rozširenia

Obr. 5.3: Webové okno sign menu

Obr. 5.4: Webové okno verify menu

- **Text to Sign**, text na podpísanie - sem vložíme text, ktorý chceme podpísať.
- **Signature Type**, typ podpisu - typ podpisu, ktorý chceme spraviť. Zatiaľ je len podporovaný typ SIG_EP.
- **Sign**, podpísať - tlačidlo na odoslanie formulára

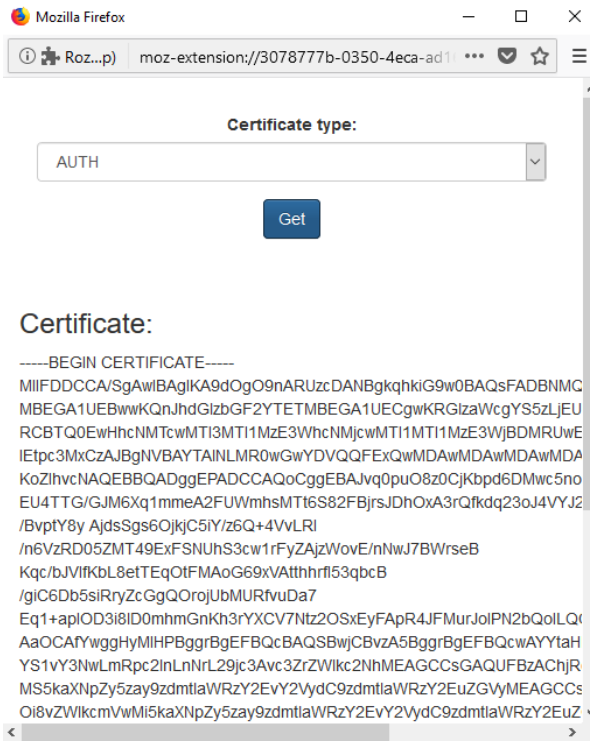
Užívateľ môže napísať alebo vložiť text do okienka **Text to Sign**, ktorý chce podpísať. Ešte pred podpísaním zvolí typ podpisu z okna možností **Signature Type** viď obrázok 5.3. Nakoniec klikne na tlačidlo a odošle text na podpísanie daným podpisom. Ak webové rozšírenie komunikuje s aplikáciu prvý krát od naštartovania Firefoxu, tak aplikácia vyhodí virtuálnu klávesnicu na zadanie BOK kódu. Nakoniec sa pod nadpisom **Signature** objaví podpis vo formáte BASE64.

2. **Verify** - overenie podpisu

   Po kliknutí na tlačidlo verify sa nám otvorí nové okno s formulárom (viď. obr. 5.4).

   Vo formulári budú tieto objekty:

   - **Text to Verify**, text na overenie - sem vložíme text, ktorý chceme overiť.
   - **Signature**, podpis - okno, kde vložíme podpis BASE64 formáte.
   - **Certificate**, certifikát - okno pre vloženie certifikátu v PEM formáte.
   - **Verify**, overenie - tlačidlo na odoslanie formuláru.



Obr. 5.5: Webové okno certificate menu

Po odoslaní sa nám pod nadpisom **Verification** objaví **true**, ak podpis patrí k podpisovanému textu a certifikát obsahuje verejný kľuč, ktorý patrí k súkromnému kľúču, s ktorým bol daný podpis vytvorený. V iných prípadoch sa nám tam zjaví **False**.

3. **Show cert** - ukázať certifikát

   Ak klikneme na tlačidlo **Show cert**, tak sa nám otvorí nové okno s formulárom. Vo formulári bude dropdown menu, kde si môžeme vybrať certifikát, ktorý chceme vybrať z čítačky. Výstup bude vo formáte PEM[3] (viď. obr. 5.5).

5.5 Overenie správnej funkčnosti aplikácie

Pred tým, ako začneme využívať aplikáciu, si môžeme overiť jej správnu funkčnosť pomocou nasledovného postupu. Ak pri nasledovnom overovaní funkčnosti aplikácie sa nám niečo nebude správať ako je napísané v postupe, odporúčame odištalovať aplikáciu a následne ju manuálne inštalovať (odkaz na postup).

[3]Možné parsovanie a vypísanie atribútov je ponechané na ďalší vývoj aplikácie.

5.5.1 Pripojenie potrebných zariadení

Overíme, či máme pripojenú čítačku, do ktorej je vložený OP s čipom. Ak nie, pripojíme ju k počítaču.

5.5.2 Vytvorenie podpisu

Otvoríme okno Sign a vytvoríme podpis na nejaký text, ktorý si sami zvolíme. Vyberieme podpis typu SIG_EP a odošleme formulár. Ak komunikujeme cez aplikáciu s čítačkou prvý krát, tak budeme musieť zadať BOK kód cez VirtualKeyboard pre autorizáciu. Nakoniec by sa nám mal zjaviť podpis BASE64 formáte (viď. obr. z práce s aplikáciu 5.3). Okno necháme otvorené pre neskoršie použitie.

5.5.3 Získanie certifikátu

Na overenie podpisu budeme potrebovať certifikát. Ak náš text je podpísaný súkromným kľúčom uloženom na OP vloženom v čítačke, tak stačí exportovať certifikát, ktorý obsahuje verejný kľuč prislúchajúci danému súkromnému kľúču, ktorým bol daný podpis vytvorený. Inak musíme vložiť iný certifikát, ktorý obsahuje ten prislúchajúci verejný kľúč. Začneme kliknutím na tlačidlo **Show cert**, ktoré nám otvorí nové okno. V tomto okne vyberieme z možností SIG_EP a klikneme na tlačidlo **Get**. Nakoniec by sa nám mal vrátiť certifikát v PEM formáte[4].

5.5.4 Overenie podpisu pomocou certifikátu

Poslednou vecou, ktorá nám zostáva, je len overiť podpis pomocou certifikátu. Na to klikneme na verify z aplikačného menu. Otvorí sa nám nový formulár. Do tohto formuláru vložíme podpísaný text, podpis a certifikát nasledovne:

- do **Text to verify** vložíme podpísaný text z prvého okna,
- do **Signature** vložíme podpis podpísaného textu získaní v prvom okne,
- do **Certificate** vložíme certifikát získaný v druhom webovom okne.

Aplikácia by mala vrátiť na výstup **true**, viď Verify (overenie podpisu). Ak nie, musíme overiť, či sme správne prekopírovali objekty alebo sme správne nainštalovali aplikáciu, či webové rozšírenie.

[4]Keby sme chceli vidieť údaje na certifikáte môžeme využiť aplikáciu OpenSSL alebo web stránku https://www.sslshopper.com/certificate-decoder.html

5.6 Ukončenie aplikácie a zhrnutie

Ak už nepotrebujeme pracovať s aplikáciou, uzavrieme všetky okná, ktoré nám aplikácia otvorila počas jej používania. Aplikácia sa sama úplne vypne pri zatvorení webového prehliadača Mozilli Firefox.

V tejto kapitole sme si prešli rôzne možnosti inštalovania pod Linuxom a OS Windows. Automatické inštalovanie funguje len v niektorých prípadoch, pretože na pokrytie všetkých možností nebol dostatok času, preto sme si popísali manuálnu inštaláciu, ktorá by mala fungovať ako pod OS Windows od verzie 7 po aktuálnu verziu OS Windows 10 a Linuxom vo väčšine známych distribúcií. Nakoniec sme si prešli, ako presne pracovať s našim riešením a ako otestovať jeho správnu funkcionalitu.

Kapitola 6

Perspektíva riešenia

V predošlých kapitolách sme sa všeobecne pozreli, ako by nám pomohli OP v našom univerzitnom systéme. Nakoniec sme sa rozhodli navrhnúť riešenie, ktoré bude mať širší záber použitia, nielen náš univerzitný systém, preto sa pozrieme na možné vylepšenia alebo zmeny v návrhu nášho riešenia. Pozrieme sa aj na nevýhody a rozdiskutujeme ako by sa dali odstrániť prípadne opraviť.

6.1 Možné vylepšenia

6.1.1 .NET CORE

Keďže aplikácia má bežať multiplatformovo a nie každý užívateľ ma nainštalovaný .NET framwork na svojom počítači, ktorý naša aplikácia vyžaduje, aby mohlo bežať. Mohlo by byť zaujímavé, presunúť projekt do .NET CORE. Týmto by sme zjednodušili užívateľovi inštaláciu aplikácie. Linuxový užívateľ by si nemusel predinštalovať Mono alebo Windows používateľ inštalovať .NET framwork, pretože pomocou .NET CORE vieme generovať native kód pre daný operačný systém. Navyše treba skontrolovať, či všetky použité moduly ako PKCS11Interop a OWIN vedia bežať pod .NET CORE. Pravdepodobne tu by nemal vzniknúť problém, pretože oba moduly už pod .NET framwork bežia multiplatformovo.

6.1.2 Inštalátory

Ak chceme, aby naša aplikácia v budúcnosti bola ľahko použiteľná, tak v prvom rade sa musí dať používateľovi bez námahy nainštalovať. Naše ponúknuté inštalátory sú len ukážkou a neriešia problém celkovo. Inštalátor pre OS Windows bol naprogramovaný pomocou Wix toolset[1]. Tento modul nám umožnil ho rýchlo spraviť, ale nerieši konfiguráciu samostatnej aplikácie. Problém spočíva v tom, že inštalátor dokáže overiť, či sú

[1] `http://wixtoolset.org`

nainštalované nutné aplikácie, ale tažko sa programuje nastavenie kľúča path pomocou inštalátora v súbore config.ini.

Podobný problém nastáva aj pod Linuxom. Naviac, Linuxový inštalátor nepodporuje overenie, či je nainštalovaný eID klient a Mono. Tu by bolo zaujímavé sa pozrieť, či nemôže tento inštalátor spustiť nejaký skript na pozadí a zistiť, či sú potrebné moduly nainštalované a ak nie, nainštalovať ich (pozn. Keď aplikácia bude bežať pod NET CORE nebude treba riešiť overovanie Mono alebo NET framwork).

6.1.3 SAML podpora

V našom riešení sa zaujímame hlavne o využitie elektronického podpisu - lokálna aplikácia vytvorí alebo overí podpis. Ďalšou možnosťou by bolo využitie autentifikácie pomocou občianskeho preukazu s čipom. Jednou z možností je využite SAML štandardu kvôli jeho rozšírenosti medzi ostatnými aplikáciami ako je Cosign[2] alebo shibboleth[3], ktoré naša univerzita využíva.

6.1.4 Vlastný PKCS modul

Naše riešenie je úplne závislé od aplikácie eID klient, čo prináša veľa nevýhod.

- Nutná predinštalácia eID aplikácie.
- Závislosť budúceho vývoja našej aplikácie od návrhu eID klienta.
- Viazanosť na licenčné podmienky.

Toto sú jedny z hlavných nevýhod, ktoré prináša využitie eID klienta. Jednoduchým riešením je použiť iný modul, ktorý bude podporovať zobrazenie Virtuálnej klávesnice alebo niečo podobné a zároveň je open-source.

6.1.5 Webové prehliadače

Naše webové rozšírenie podporuje iba Firefox, čo v celkovom obraze obmedzuje viacerých užívateľov, pretože užívatelia používajú aj iné prehliadače ako sú Google Chrome, Opera, Safari. Predtým než začneme prenášať náš kód, treba si overiť, či daný webový prehliadač podporuje native messaging a iné funkcionality, ktoré využíva naše webové rozšírenie. Taký Google Chrome podporuje všetky využité funkcionality a preto implementácia webového rozšírenia bude jednoduchá. Stačí, ak človek prejde všetky nekompatibilné časti v našom rozšírení. Tie nájde na stránke `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Chrome_incompatibilities`.

[2] `http://weblogin.org`
[3] `https://www.shibboleth.net`

6.1.6 Automatické podpisovanie

Jednou z plánovaných výhod aplikácie bude využitie elektronického podpisu na internete. Problémom však zostáva automatizácia riešenia. Užívateľ sám musí interagovať s webovým rozšírením aj vtedy, kedy by to nebolo treba.

- Podpisovanie súboru po procese nahrávania súboru.

  Po nahratí súboru sa užívateľovi vyhodí okienko, kde si užívateľ môže zvoliť, či tento súbor chce podpísať. Ak áno, tak aplikácia stiahne súbor a nahrá ho znova, ale už podpísaný.

- Podpisovanie textu na webovej stránke.

  Užívateľ bude môcť označiť text na stránke. Následne po pravom kliknutí myšou sa mu zjaví aj možnosť popísania textu. Ak zvolí túto možnosť, webové rozšírenie podpíše text a podpis vráti v novom okne aj s podpisovaným textom a patričným certifikátom.

- Vloženie certifikátu do textového okna alebo do stránky.

  Ďalšou možnosťou môže byť, že po kliknutí pravým tlačidlom na myške sa zjaví menu, kde si užívateľ bude môcť vybrať, aký certifikát chce a v akom formáte. Následne aplikácia môže vložiť certifikát do textového okna, na ktoré užívateľ ukazuje myškou alebo otvorí nové okno s týmto certifikátom.

- Automatické overenie podpisu.

  Ak webové rozšírenie nájde elektronický podpis s certifikátom a text alebo súbor, tak automaticky to overí a môže podfarbiť alebo inak znázorniť výsledok overenia.

- Získane informácie z certifikátu v PEM formáte.

  Webové rozšírenie bude umožňovať po kliknutí na certifikát v PEM formáte výpis jeho obsahu v čitetelnej forme v novom okne.

Je množstvo možností, ktoré by sa dali automatizovať. My sme iba načrtli niektoré z nich, ktoré by bolo vhodné implementovať.

6.1.7 Podpora bezpečnej komunikácie

Celá komunikácia medzi lokálnou aplikáciou a webovým rozšírením nespĺňa ani jednu bezpečnostnú požiadavku, preto ďalšou možnosťou vylepšenia nášho riešenia by bolo aspoň na základnej úrovni spĺňať integritu a dôvernosť.

Jednou z možností, ako zabezpečiť integritu prenosu dát je použiť HMAC[4]. Tuto nastáva problém s uchovaním kľúča alebo s jeho generovaním (Niekto môže odpočúvať na sieti.). Jedno z možných riešení tohto problému by vyzeralo nasledovne:

1. Pred spustením lokálnej aplikácie by webové rozšírenie vygenerovalo náhodne tajný kľuč, ktorý sa bezpečne nazdiela medzi webovým rozšírením a lokálnou aplikáciou.

2. Následne by spustil lokálnu aplikáciu a cez native messaging by poslal tento kľuč.

3. Lokálna aplikácia by prijala kľuč a celá serverová komunikácia by prebehala s použitím tohto kľúča

Tento návrh by zabezpečil nejakú ochranu integrity a dôvernosť dát medzi lokálnou aplikáciou a webovým rozšírením. (pozn. Tento návrh len ukazuje, akým smerom by sa to dalo možno riešiť, neukazuje to celkové riešenie problému.)

6.2 Otázky

6.2.1 Bezpečnosť riešenia

V tejto časti sa pozrieme na bezpečnosť našej aplikácie. Pozrieme sa aj na to, v ktorých prípadoch sa dá zabrániť útokom a v ktorých sa nedá.

Naše navrhované riešenie z časti Podpora bezpečnej komunikácie 6.1.7 funguje iba na sieťovej vrstve. Akonáhle má útočník prístup k všetkým aplikáciám ako normálny užívateľ, tak by dokázal odpočúvať štandardný vstup a výstup, na ktorom beží native messaging. V našom riešení by vedel zistiť predzdieľané heslo, čo by spôsobilo, že komunikácia by už nevedela zaručiť integritu dát.

Lokálna aplikácia vyžaduje heslo iba na prvý prístup k občianskemu preukazu a pri podpisovaní elektronickým podpisom. Toto by vedel zneužiť útočník nasledovne:

1. Bude odpočúvať na sieti a čakať na prvú interakciu s lokálnou aplikáciou od užívateľa.

2. Následne, ak deteguje prvú interakciu, vytvorí vlastnú požiadavku pre podpísanie vlastného hashu.

Lokálna aplikácia pri prijatí druhého požiadavku už nevyhodí okno pre zadanie BOK kódu. Týmto spôsobom môže útočník elektronicky podpísať ľubovoľné dáta bez toho, aby o tom užívateľ vedel.

Za úvahu by stálo, či vždy pred vytváraním kvalifikovaného elektronického podpisu bude modul od uživatela vyžadovať zadanie ZEP kódu.

[4] `https://sk.wikipedia.org/wiki/HMAC`

6.2.2 Komunikácia s inými aplikáciami

Náš návrh nie je veľmi flexibilná. Komunikácia je navrhnutá iba medzi webovým rozšírením a lokálnou aplikáciou. Možným rozšírením tohto konceptu by bolo, keby lokálna aplikácia definovala alebo využívala už existujúci komunikačný protokol, čím by umožnila aj iným aplikáciám využívať občiansky preukaz s čipom.

Jednou z možností sú lokálny emailoví klienti ako je Outlook, ktorí by mohli podpisovať emaily alebo využiť možnosť autentifikácie na prístup k emailom využitím občianskeho preukazu s čipom.

Ďalšou možnosťou by bolo využitie samotného operačného systému, ktorý by mohol vylepšiť bezpečnosť našej aplikácie. Žiaľ, toto by sa muselo implementovať na každý operačný systém samostatne, čo by prinieslo problémy pri ďalšom vývoji aplikácie.

6.3 Zhrnutie

Naše riešenie prešlo dlhým vývojom, aby sa dostalo k tejto ukážkovej verzii, ale ešte stále nie je v dostatočnej verzii na implementovanie do nášho univerzitného systému. V tejto kapitole sme sa pozreli na jednotlivé môžnosti vylepšenia nášho riešenia a zodpovedali otázky, ktoré sú nutne zodpovedať a implementovať ich odpovede do riešenia ešte pred dokončením finálnej verzie.

Záver

V našej práci sme navrhli a popísali riešenie, ktoré umožňuje dvojfaktorovú autentifikáciu do univerzitného systému alebo podpisovanie a šifrovanie e-mailov a dokumentov. Pretože väčšina užívateľov používa webový prehliadač, naše riešenie sa zameralo na implementovanie týchto vlastností doňho. Umožní nám to v budúcnosti ľahšie vyvíjať našu aplikáciu pre webové portály alebo iné verejné správy.

Nakoľko návrh je veľmi rozsiahli na bakalársku prácu, implementovali sme len náznakové riešenie, ktoré sa skladá z webového rozšírenia a lokálnej aplikácie. Webové rozšírenie slúži na komunikáciu s užívateľom a umožňujeme vytvoriť elektronický podpis z daného textu, získať certifikát v PEM formáte a overenie elektronického podpisu. Avšak webové rozšírenie nemohlo priamo komunikovať s PKCS knižnicou kvôli implementačným nastaveniam Mozilli Firefox, preto sme navrhli a naprogramovali lokálnu aplikáciu, ktorá komunikuje tak s webovým rozšírením, ako aj s PKCS knižnicou. Použitím tejto knižnice dokáže vytvoriť elektronický podpis alebo získať certifikáty a následne ich poslať webovému rozšíreniu ako odpoveď na požiadavku, ktorú webové rozšírenie vygenerovalo pri interakcii s užívateľom.

Pri vyvíjaní lokálnej aplikácie sme narazili na množstvo problémov. Jedným z nich bol výber programovacieho jazyka. Prvým návrhom bol Python, no po zistení jeho malej podpory knižníc ku PKCS štandardu sme sa rozhodli prejsť do jazyka C# a využitie knižníc ako pkcs11interop a OWIN (v jazyku C#) nám umožnilo abstrahovať od implementačných detailov. Naprogramovanie vlastnej PKCS knižnice je veľmi náročné a presahovalo by rámec bakalárskej prace. Rozhodli sme sa preto použiť knižnicu z eID klient programu, ktorý štát vydal na komunikáciu s verejnou správou.

Nakoniec sme si popísali navrhované riešenie. Načrtli sme, ako prebieha komunikácia medzi jednotlivými modulmi pomocou požiadaviek, staticky sme opísali jednotlivé moduly a prečo sme zvolili daný návrh tried a štruktúr.

V poslednej kapitole sme rozdiskutovali možné vylepšenia nášho riešenia a ako by sa jednotlivé problémy dali odstrániť. V tejto časti sme sa zamerali aj na bezpečnosť komunikácie medzi jednotlivými modulmi nášho dema. Zistili sme, že obsahuje nesmierne nedostatky a k niektorým z nich sme navrhli riešenia, ktoré by mohli viesť k vyššej bezpečnosti.

Naše demo riešenie ešte nie je plno využiteľné pre dvojfaktorovú autentifikáciu alebo

automatické podpisovanie emailov, ale ukazuje, ktorým smerom by mal ísť ďalším vývoj v tomto smere, čo umožní ďalším vývojárom preskočiť prevodné problémy, s ktorými sme sa potýkali pri navrhovaní systému, a môžu sa rovno zamerať na jednotlivé aplikácie, čo prinesie želané efektívne a bezpečné riešenie.

Literatúra

[1] Pkcs #11 cryptographic token interface base specification version 2.40. http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html.

[2] Zákon č. 215/2002 o elektronickom podpise a o zmene a doplnení niektorých zákonov.

[3] Zákon č. 224/2006 o občianskych preukazoch a o zmene a doplnení niektorých zákonov.

[4] Cryptographic hash function. https://en.wikipedia.org/wiki/Cryptographic_hash_functionl, Apríl 22, 2018.

[5] Daniel Olejár. *Stručný výkladový slovník informačnej a kybernetickej bezpečnosti.* 2016.

[6] William Stallings, Lawrie Brown, Michael D Bauer, and Arup Kumar Bhattacharjee. *Computer security: principles and practice.* Pearson Education, 2012.

[7] Douglas R Stinson. *Cryptography: theory and practice.* CRC press, 2005.

Zdrojový kód demo riešenia

Táto príloha obsahuje inštalátory a zdrojové kódy nášho riešenia na komunikáciu medzi občianskym preukazom s čipom a webovým rozšírením, ktoré sme si priblížili v kapitole 4 a prebrali ich inštaláciu v kapitole 5.

Zdrojové kódy a inštalátory nájdeme v súbore **riesenie.zip** na priloženom CD. Pre lepšiu orientáciu v prílohe sme vytvorili súbor **ReadMe.txt**.