VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Matěj Miška

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

ELEKTRONICKÝ PODPIS V PRAXI
ELECTRONIC SIGNATURE IN PRACTICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Matěj Miška

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Vlastimil Člupek, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Matěj Miška ***ID:*** 200456
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Elektronický podpis v praxi

POKYNY PRO VYPRACOVÁNÍ:

V bakalářské práci proveďte analýzu možností tvorby digitálních podpisů, získání digitálních certifikátů a bezpečného uložení podpisových klíčů. Otestujte běžně využívané způsoby podpisu elektronických dat (emailů, PDF dokumentů apod.) pomocí existujících aplikací a určete jejich výhody a nevýhody. Navrhněte zabezpečenou aplikaci pro správu digitálních certifikátů a podpisových klíčů, jenž bude umožňovat podepisování PDF dokumentů a libovolných elektronických dat. Vytvořte Vámi navrženou zabezpečenou aplikaci, ověřte její funkčnost, popište její přednosti a přehledně prezentujte možnosti jejího využití.

DOPORUČENÁ LITERATURA:

[1] BUDIŠ, Petr. Elektronický podpis a jeho aplikace v praxi. ANAG, 2008.

[2] RIVEST, Ronald L.; SHAMIR, Adi; ADLEMAN, Leonard. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978, 21.2: 120-126.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. Vlastimil Člupek, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá tvorbou systému pro správu a zabezpečení elektronických certifikátů určených k podpisu dokumentů. Aplikace vytvořená v této práci je koncipována jako konzolová aplikace volána z příkazové řádky systému. Toto řešení je odůvodněno možností pokračování této práce v podobě nástavby uživatelského rozhraní nebo využití jinou aplikací. Pomocí tohoto programu je možné vytvářet uživatele, kteří mohou nahrávat své certifikáty s hesly do aplikace a následně je využít při podpisu souborů PDF a PNG. V aplikace jsou implementovány algoritmy zajišťující bezpečnost uložených dat. Teoretická část práce podrobně rozebírá tvorbu a práci s certifikáty a podpisovými klíči v reálném světě včetně tvorby vlastního self-signed certifikátu. Závěrem teoretické části je testování využitelnosti certifikátů.

KLÍČOVÁ SLOVA

uživatel, certifikát, podpis, úložiště, X.509, zabezpečení, C#, AES, hash

ABSTRACT

This bachelor's thesis deals with the creation and development of a system for management and security of digital certificates used for signing documents. The application created in this thesis is conceptualised as a console application used from command line of a system. This solution is justified by the possibility of being continued as a graphic user interface extension or adopted by another application. The application allows for the creation of a user account, through which a person can import their certificates with passwords into the application and subsequently use the certificates in signing PDF documents or PNG files. Algorithms ensuring the security of saved data are implemented by the application. The theoretical part of this thesis analyzes the creation and work with certificates and signature keys in real life, including the creation of own self-signed certificate. The theoretical part concludes with the testing of usage of the certificates.

KEYWORDS

user, certificate, signature, storage, X.509, security, C#, AES, hash

MIŠKA, Matěj. *Elektronický podpis v praxi*. Brno, Rok, 57 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Vlastimil Člupek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Elektronický podpis v praxi“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Vlastimilu Člupkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Elektronický podpis se stává fenoménem moderní doby a díky své využitelnosti a právnímu zakotvení se staví na stejnou úroveň jako klasický podpis. Oblast internetu a zjednodušování života jakoukoliv formou představuje v dnešní době lákavý cíl. Provádění důležitých rozhodnutí přes internet, jako například internetové bankovnictví či podepisování smluv, čelí ovšem velké překážce, kterou je autentizace.

Autentizací je myšlen proces identifikace a ověření podepsané osoby. Vzhledem k tomu, že elektronický podpis je „nehmotný“ a jedná se pouze o data, existuje zde větší množství způsobů falzifikace a zneužití, než u podpisu klasického.

V poslední době si tedy můžeme všímat snahy vyvinout co nejspolehlivější algoritmy pro tvorbu těchto autentizačních prvků zvyšujících důvěryhodnost digitálních podpisů. Tato práce bude pojednávat o tvorbě, uchovávání a autentizaci entit využívajících tyto podpisy.

V zásadě se dá za podpis a podepsání se na internetu brát většina operací, při kterých se určitá osoba identifikuje a zadává identifikační údaje, které má výlučně ve svém vlastnictví. Z pohledu legislativy se však jedná o různé případy s různými právními dopady.

Těchto podpisů tedy existuje více druhů s rozdílnou právní silou a rozdílnými požadavky na tvorbu. Z výše zmíněných důvodů se následující práce bude primárně zaměřovat na tzv. zaručený elektronický podpis, který je uznávaný státními orgány a je tudíž považován za standart v oblasti elektronického podpisu.

1 Typy elektronických podpisů

První část této práce definuje elektronický podpis a rozvádí ho do několika příkladů jeho implementace. Hlavním objektem zkoumání je především zaručený elektronický podpis, který je možné považovat za základní formu legitimního elektronického podpisu využitelného v praxi díky jeho právní síle a zakotvení v právním řádu. Podklady k této kapitole byly získány ze zdrojů [1, 2].

1.1 Definice elektronického podpisu

Elektronický podpis bohužel není jednoduše definovatelný jednou větou. Existuje několik typů elektronického podpisu s rozdílnou kvalitou a rozdílnými nároky na autentizaci, z čehož zároveň vyplývá, že tyto podpisy mají také rozdílné využití a hodnotu. Pro lepší pochopení si lze elektronický podpis představit jako velmi dlouhý řetězec znaků (v praxi je doporučeno využívat klíče o minimální délce 2048 bitů kvůli bezpečnosti), který po připojení k dokumentu dokáže autora nejen identifikovat, ale také určit čas podpisu, a zda podepsaný dokument nebyl v době od svého podpisu pozměněn, nebo dokonce zda se za osobou podepsanou pod tímto dokumentem skutečně nachází stejná osoba.

K tomu, aby podpis mohl prokázat veškeré tyto klíčové údaje, je potřeba k řetězci znaků připojit také další validační prvky. Celý podpis je založený na asymetrické kryptografii, která vyžaduje k vytvoření a ověření podpisu dva rozdílné klíče (data). Ty nelze navzájem od sebe odvodit, aby soukromý klíč využívaný k podpisu dokumentu zůstal ve výhradním vlastnictví majitele.

1.2 Zaručený elektronický podpis

Tento typ digitálního podpisu je nejrozšířenější a obecně nejvyužívanější z důvodu jeho kvality, záruky a nízké časové náročnosti jeho ověření. Pro praktičnost a jednodušší srozumitelnost si tato práce bere zaručený elektronický podpis jako standard, na kterém bude popisovat základní principy tvorby a využití těchto podpisů. Jeho využití se vztahuje primárně na podepisování smluv, korespondenci se státní správou, komunikaci se soudy nebo právnickými osobami. Z těchto důvodů má elektronický podpis nejblíže ke standardnímu podpisu fyzické osoby.

Zaručený elektronický podpis se skládá ze samotného podpisu, časového razítka a certifikátu o zaručeném elektronickém podpisu, vydaného autoritou potvrzující majitele podpisu, čímž stvrzují jeho pravost. Díky všem těmto částem, které digitální podpis obsahuje, ho lze považovat za právně silnější, než klasická verze podpisu. Důvodem k tomuto tvrzení je nepopiratelnost, kterou podpis zaštiťuje, tedy že osoba

identifikovaná pod tímto podpisem nemůže popřít, že daný podpis vytvořila (resp. nemůže odmítnout důsledky vyplývající ze stvrzení tohoto dokumentu), jelikož jediná osoba držící data potřebná k tvorbě a podepsání dokumentu je její majitel.

1.3 Elektronická značka

Výše zmiňovaný zaručený elektronický podpis má pouze jediné omezení, a to skutečnost, že ho může vlastnit pouze fyzická osoba. Pokud se chce tedy s pomocí elektronického podpisu identifikovat osoba právnická či organizační složka státu, musí využít právě elektronickou značku. Po technické stránce se s ní jedná jako se zaručeným elektronickým podpisem. Ke vzniku této značky může dát příkaz i stroj nebo program bez účasti člověka, ovšem právní odpovědnost za stroj nese člověk, který stroj nastavil k vytváření značek (tzv. označující osoba).

1.4 Časové razítko

Speciálním typem elektronického podpisu je časové razítko. Podává detailní informaci o čase stvrzení dokumentu, a tím podpis obohacuje oproti jeho obvyklé verzi. Při validaci tedy jakákoliv osoba kontrolující dokument dokáže určit stáří podpisu a také to, zda byl podpis v době přiložení k dokumentu aktuálně platný. V praxi se nejčastěji používají tzv. kvalifikovaná časová razítka, která podobně jako zaručený elektronický podpis zaručují správnost údajů díky tomu, že jsou vytvářena kvalifikovaným uživatelem.

1.5 Biometrický podpis

Biometrický podpis je velmi odlišný od klasického elektronického podpisu, mají však společnou autentizační vlastnost. Tento podpis je založen na jakékoliv biometrické charakteristice člověka, tedy například snímku sítnice, otisku prstu, nebo jakým způsobem člověk píše svůj klasický podpis, čímž zajišťuje daleko větší bezpečnost v procesu samotné tvorby podpisu. Výsledek tohoto podpisu by totiž měl být naprosto unikátní pro každého jedince. Tato originalita by měla zajistit neomezenou platnost z důvodů těžké napodobitelnosti a nepřenositelnosti. Problémů spojených s biometrickým podpisem je však hned několik. Například jeho uchovávání, distribuce, nebo možnost využití pouze fyzickou osobou.

2 Certifikát elektronického podpisu

Tato kapitola popisuje certifikáty vydané certifikačními autoritami, princip využití certifikátů a proces jejich získání [1].

2.1 Využití certifikátů

Vzhledem k tomu, že je elektronický podpis pouze řetězec dat, je nutné, aby se daný podpis dal prokazatelně spojit s jeho autorem. K tomuto účelu se využívají certifikáty. Certifikáty se přikládají k podpisům, aby stvrzovaly jejich platnost, platnost přiloženého veřejného klíče a spojily podpis s osobou vlastnící soukromý klíč nutný k vytvoření podpisu. Obdobně jako u podpisu, certifikáty se dělí na několik druhů. Například na osobní, které může vlastnit pouze fyzická osoba, a certifikáty systémové, které jsou využívány organizačními složkami státu, případně se pomocí nich vytváří časová razítka, šifrují servery nebo komunikace v rámci SSL relací.

Při dělení certifikátů dle jiných kritérií je můžeme dále rozdělit na kvalifikované certifikáty, jejichž obsah je vymezen v zákoně, a na komerční, jejichž obsah se může lišit.

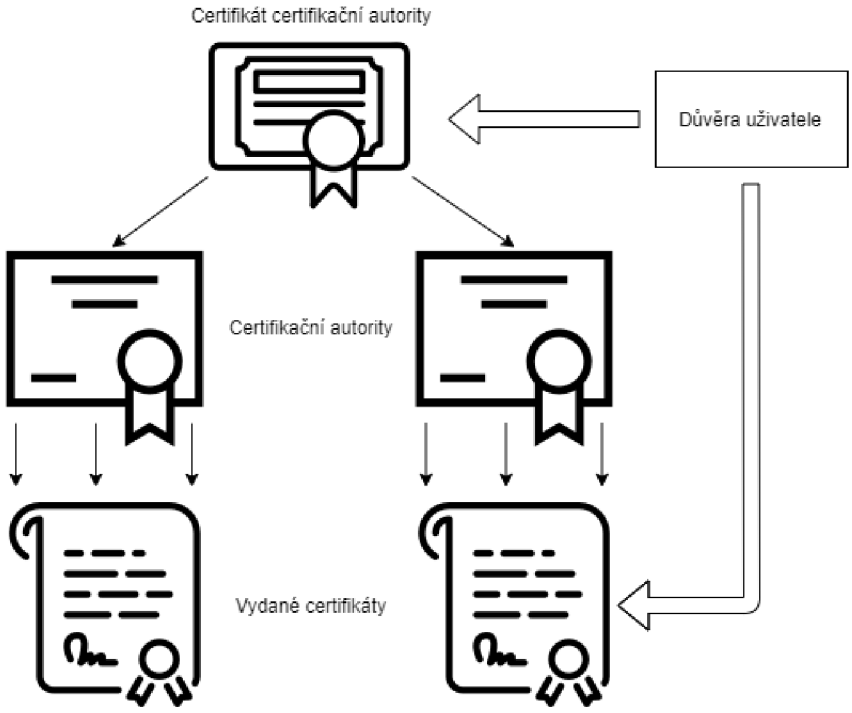
2.1.1 Certifikační autority

Certifikační autorita je organizace, která na požádání vydává certifikáty k veřejným klíčům (v právnické terminologii se jedná o poskytovatele certifikačních služeb). Touto autoritou může být kdokoliv, ovšem pro maximální možnou důvěryhodnost podpisu se využívají certifikáty státem schválených organizací. Na území České republiky se jedná o První certifikační autoritu (I. CA), eIdentity a PostSignum. Tyto organizace vydávají jak kvalifikované, tak i komerční certifikáty k veřejným klíčům na omezenou dobu, aby nedošlo ke zneužití celého podpisu i s jeho komponenty.

2.1.2 Infrastruktura veřejného klíče (PKI)

Kromě vydávání certifikátů mají certifikační autority také na starost ověřování vydaných certifikátů pomocí PKI (Public key infrastructure), která se zakládá na důvěře v kořenový certifikát. Pokud tedy kontrolující osoba věří tomuto certifikátu, je potom i certifikát vydaný touto organizací považován za platný. Toto přenesení důvěry se dá opakovat na dalším certifikátu vytvořeném na základě předešlého, čímž vznikají celé „stromy důvěry“, jak je možné vidět v obr. 2.1.

Základem je tedy kořenový certifikát, na který se váží certifikáty z něj vytvořené, a který vlastní certifikační autorita, tudíž je mu možné důvěřovat. V praxi se



Obr. 2.1: Teoretická ukázka stromu důvěry.

však využívá více než jen jedné certifikační autority, čímž vzniká vícero stromů důvěry. Splynutím několika stromů tedy vzniká infrastruktura veřejného klíče, jejímž úkolem je zajistit důvěryhodný systém pro ověřování veřejných klíčů obsažených v certifikátech pro validaci podpisu.

2.2 Proces získání certifikátu

Pro získání certifikátu je nutné vytvořit žádost o vydání certifikátu. Žádostí se rozumí datový soubor ve formátu PKCS#10, obvykle s příponou „.req“. S vytvořením žádosti souvisí také vygenerování a připojení nezbytných párových dat, které jsou následně zaslány společně se žádostí. Po vyhotovení žádosti se tento datový soubor posílá certifikační autoritě. Je nutné se však předem rozhodnout k jakému účelu bude podpis sloužit, jelikož certifikační autorita neodpovídá za případné chyby (jako například vygenerování neexportovatelného klíče).

Již zmíněné žádosti je možné generovat několika způsoby. Nejjednodušším způsobem generování pro běžného uživatele je generování „on-line“, které se provádí pomocí běžného webového prohlížeče. Druhou variantou vhodnější pro specialistu podepisujícího dokumenty na denní bázi jsou „off-line“ generátory využívající stažitelnou aplikaci.

2.2.1 On-line způsob

Generování žádosti on-line způsobem vyžaduje, aby webový prohlížeč, který bude žádost generovat, podporoval příslušnou certifikační autoritu a považoval ji za důvěryhodnou. Tato důvěra je důležitá z důvodu, že uživatelův prohlížeč zasahuje do počítače, kde v příslušném úložišti certifikátů generuje párová data. „Považování za důvěryhodnou“ znamená považovat za důvěryhodný především kořenový certifikát dané autority, díky čemuž bude prohlížeč považovat za důvěryhodný i serverový certifikát, kterým se prokazují webové stránky autority. Různé webové prohlížeče a certifikační autority mohou vyžadovat dodatečné plug-iny, které doplňují prohlížeč o spustitelné kódy pro sestavení certifikátů.

Po zadání URL adresy a připojení se na webovou stránku následuje vyplnění údajů pro generování žádosti. Po uživateli jsou vyžadovány osobní údaje společně s požadovanou délkou klíče a požadovaným umístěním úložiště. Na zvolení úložiště zároveň závisí poskytovatel kryptografických služeb, který generuje soukromý klíč v příslušném úložišti.

2.2.2 Off-line způsob

Tento způsob generování žádostí se využívá převážně pokud webové prohlížeče uživatele nejsou podporovány certifikační autoritou, nebo pokud se jedná o generovaní většího počtu žádostí. V aplikaci jsou také implementované funkcionality, které je v případě on-line způsobu vytvoření žádosti potřeba do prohlížeče doinstalovat. Samotná aplikace funguje na stejném principu jako u on-line způsobu a po zadání veškerých potřebných údajů vygeneruje aplikace žádost o certifikát.

3 Tvorba podpisu

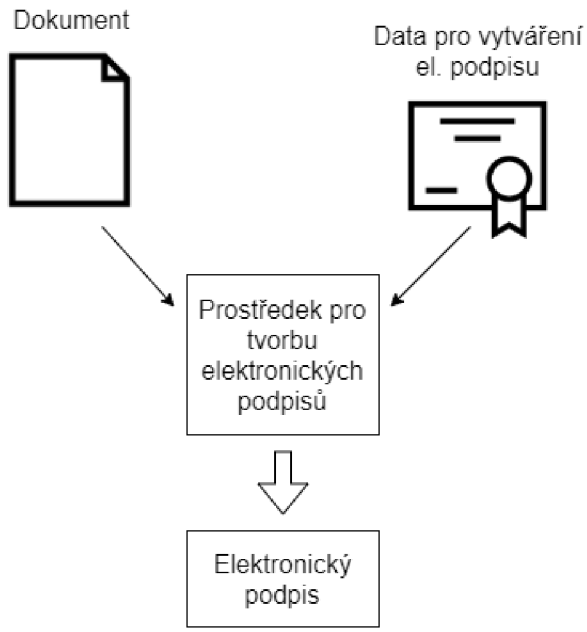
Hlavní výhodou při využívání elektronického podpisu je jeho univerzálnost a nekonečná možnost užití, pokud je jeho certifikát (a certifikáty jemu nadřazené) platný (viz [1]). Odpadá tedy nutnost podepisovat každý dokument novým podpisem, a tedy osobě postačí vždy pouze jeden platný podpis. Při tvorbě podpisu je však nutné, aby podpis zůstal originálem, a zároveň jeho tvorba byla rychlá a univerzální.

3.1 Univerzální část tvorby podpisu

Tato část tvorby je nazvána univerzální z toho důvodu, že ji tvůrce může svěřit programu. Ten je volně šiřitelný, protože nevytváří podpis originální, nýbrž pouze tu část podpisu, kterou disponuje každý podpis. Z právního hlediska se jedná o „prostředky pro vytváření elektronických podpisů“.

3.2 Individuální část tvorby podpisu

Individuální část elektronického podpisu se skládá z veřejného a soukromého klíče, využívaných při tvorbě zaručeného elektronického podpisu, díky kterým je možné vytvořit originální podpis a následně ho ověřit. V právní legislativě je tato část podpisu označována jako „data pro vytváření elektronického podpisu“, viz obr. 3.1.



Obr. 3.1: Základní prvky nutné k tvorbě elektronického podpisu.

3.3 Algoritmy používané ve spojení s elektronickým podpisem

Při tvorbě podpisu se využívá kryptografických algoritmů sloužících k utajení a šifrování dat, viz [4].

3.3.1 Algoritmus RSA

Jedním z těchto algoritmů je Rivest, Shamir, Adleman (RSA) algoritmus, vhodný jak k podepisování, tak k šifrování. Při vytváření podpisu se využívá především k tvorbě páru RSA klíčů, jejichž bezpečnost je založena na předpokladu, že rozkladu velkého čísla na součin dvou prvočísel je nemožné docílit v polynomiálním čase. Proto se v současnosti při použití dostatečně dlouhého klíče stále považuje algoritmus RSA za bezpečný.

Při podepisování se používají RSA klíče v „opačném pořadí“ a předně k ověřování pravosti podpisu. Autor podepíše zprávu svým soukromým klíčem a při ověření se využije dešifrování pomocí veřejného klíče, který je volně k dispozici. Tím se určí jediný vlastník soukromého podpisu a tedy i jeho identita.

3.3.2 Algoritmus DSA a ECDSA

Digital Signature Algorithm (DSA) je standardem americké vlády pro elektronický podpis, který byl vydán veřejnosti k volnému užívání. Celý algoritmus je založen na problému výpočtu diskrétního logaritmu a využívají ho například programy OpenSSL nebo OpenSSH. Pro správnou funkčnost je třeba vybrat hashovací funkce, v dnešní době SHA-2, a parametry L a N určující délku klíčů - nejčastěji o délce L = 2048 a N = 256.

Algoritmus ECDSA funguje na stejném principu jako DSA, pouze navíc využívá eliptických křivek ke zvolení parametrů pro vytvoření klíčů. Předpokladem pro užití eliptických křivek je nemožnost nalezení diskrétního logaritmu náhodného bodu této křivky,
i když známe její hlavní bod. Pro současné kryptografické účely se využívají eliptické křivky, které popisuje rovnice

$$y^2 = x^3 + ax + b. \tag{3.1}$$

3.3.3 Hashovací funkce

Hashovací funkce je jednosměrná kompresní funkce umožňující z libovolně dlouhého řetězce znaků vytvořit hashový otisk, který se následně využívá při tvorbě podpisu

a časového razítka. Díky své jednosměrnosti se využívá k vytváření identifikačních řetězců sloužících k ověřování (například LM Hash u Microsoft Windows XP) nebo k dalšímu zpracování, kdy je nezbytné vytvořit unikátní identifikátor datového řetězce. Hashovací funkce by také měla splňovat určitý standard, aby zamezila vzniku kolizí, a tedy vznikalo co nejméně výše zmíněných kolizních dokumentů. Vzhledem k rychlému vývoji těchto systémů se v dnešní době doporučuje využívat minimálně hashovací funkci SHA-2 společně s klíči o minimální délce 2048 bitů. Například hashový otisk tohoto odstavce pomocí SHA-2 (o délce 512 bitů) vypadá takto:

$$1937463944a5943868ebd02be6e020b1b4873cd14c35f736fc06eb90.$$

3.4 Integrita podpisu a dokumentu

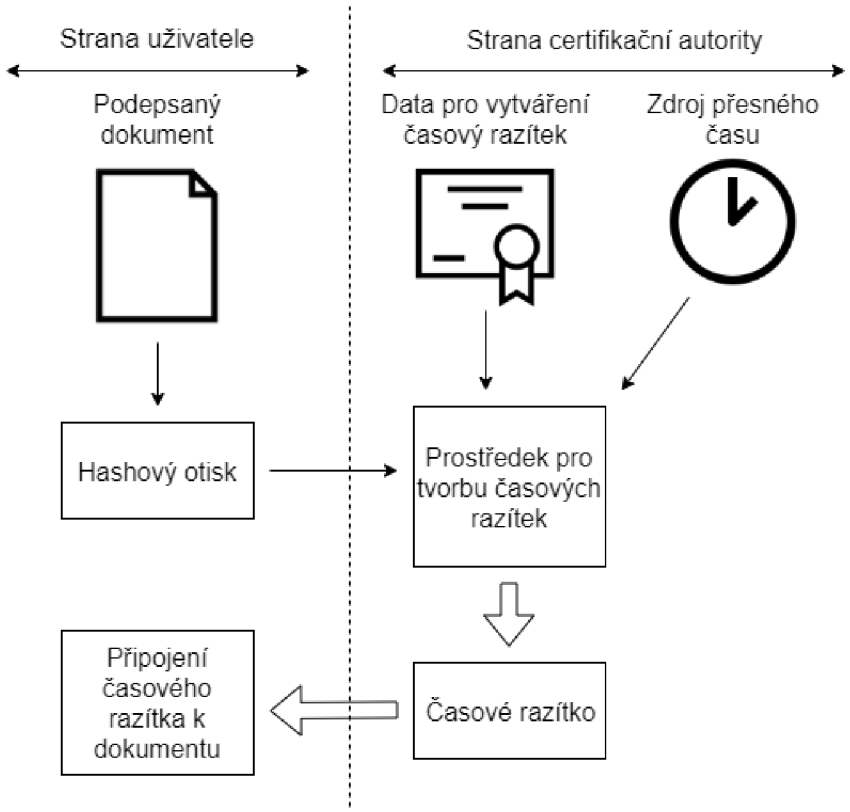
Podpis sám o sobě existovat může, ačkoliv jeho plné využití spočívá v udržení integrity dokumentu. Aby podpis získal tuto vlastnost, je třeba aby odrážel obsah dokumentu, a byla porušena vazba na dokument při jeho úpravě. Dokument se tedy přidává ke vstupu do výpočtu podpisu, který vykonává program pro výpočet elektronických podpisů, společně se soukromým klíčem.

Tento postup však vyvolává problém z důvodu podepisování se stejným podpisem na vícero dokumentů. V tomto případě mohou vznikat tzv. kolizní dokumenty. Kolizními dokumenty se označují dva a více dokumentů, které mají stejný hashový otisk, a tudíž i stejný podpis. Tomuto problému se zabraňuje časovými razítky, omezujícími platnost dokumentu. Ty však pouze oddalují možnost vzniku kolizního dokumentu, a proto je nutné tyto dokumenty a jejich časová razítka „aktualizovat“.

3.5 Omezení časové platnosti podpisu

Z důvodu neustálého vývoje dokonalejších technologií je nezbytné zabezpečit podpisy proti zestárnutí, a tedy případnému prolomení a zneužití. K tomuto účelu se přikládají k podpisu časová razítka, jelikož samotný podpis není časově omezen. Toto časové razítko neovlivňuje platnost celého dokumentu, pouze limituje možnost ověřit platnost podpisu. Musí tedy být stanoveno na základě prokazatelného časového údaje, a zároveň být vydáno důvěryhodnou třetí stranou (nejčastěji stejnou autoritou distribuující certifikáty). Platnost časového razítka se pohybuje od 3 do 6 let, kdy po uplynutí této lhůty lze podpis oživit uměle skrze nově vytvořené časové razítko, které prodlouží platnost celého dokumentu. Aby razítko potvrzovalo čas platnosti dokumentu, musí být připojeno k dokumentu až po jeho podepsání, čímž stvrzuje, že v daný okamžik byl celý dokument platný. K jeho tvorbě se využívá stejného principu jako u elektronického podpisu s tím rozdílem, že časové razítko musí

svým vlastním klíčem „podepsat“ kvalifikovaný poskytovatel certifikačních služeb a až po tomto kroku razítko nabývá své právní moci, viz obr. 3.2.



Obr. 3.2: Komponenty poskytované autoritami a uživatelem při tvorbě podpisu.

3.6 Přiložení podpisu k dokumentu

K dokumentu může být elektronický podpis přiložen interně nebo externě. V praxi se převážně využívá interní připojení podpisu k dokumentu z důvodu jeho trvalého sloučení s dokumentem. Problém však nastává při nutnosti podepsání dokumentu vícero podpisy, což se řeší zapouzdřením podpisu i s dokumentem a manipulací s ním, jako by se jednalo o celistvý dokument.

4 Ověřování elektronického podpisu

Při zkoumání platnosti podpisu mohou nastat pouze situace kdy je podpis platný, neplatný či nejsme schopni potvrdit jeho platnost (viz [2]). Nemožnost ověření podpisu neznamená automaticky jeho neplatnost, ovšem nebude na něj nahlíženo jako na podpis platný. Jediný časový údaj, který je možné zjistit o platnosti tohoto podpisu je doba stvrzení, čímž je jasné, že od této doby podpis existoval.

Z tohoto důvodu tedy není jednoduché prohlásit podpis za neplatný. Jediný případ neplatného podpisu nastává, pokud byla porušena integrita celého dokumentu. Avšak to, že integrita porušena nebyla neznamená, že je podpis platný. Obdobným procesem musí být ověřena také platnost certifikátu stvrzujícího podpis a celý certifikační strom, ze kterého daný certifikát pochází, jelikož certifikáty mohou být předčasně revokovány.

Dále se určuje legislativní síla podpisu a způsob přiložení k podepisovanému dokumentu.

4.1 Ověření integrity dokumentu

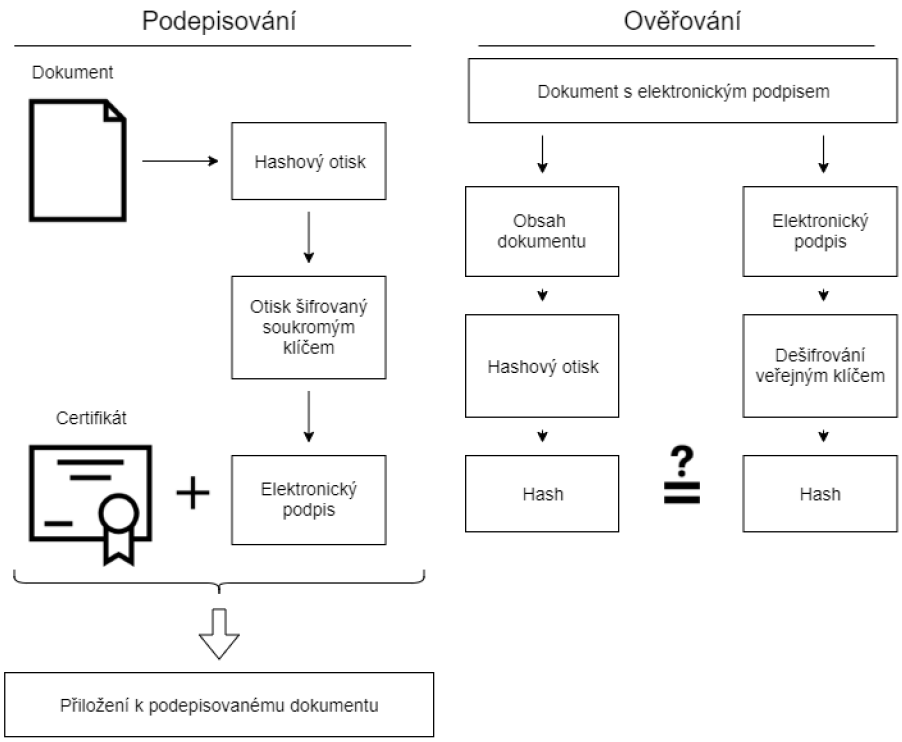
Nejjednodušší částí ověřování a vyhodnocování platnosti dokumentu je posouzení stavu jeho integrity. K tomuto ověření se využívá hashovacích funkcí, díky kterým se může porovnat hashový otisk dokumentu společně s dešifrovanými daty pomocí veřejného klíče.

Oba klíče jsou unikátní jak svým obsahem, tak využitím. Při podepisování se využije soukromého klíče jako vstupu v procesu tvorby podpisu, zatímco veřejný klíč je určen pouze k ověření a je tedy volně dostupný, viz obr. 4.1. Z popisu klíčů vyplývá, že klíče jsou jeden bez druhého bezcenné. Kontrola se uskutečňuje dopočítáním nového hashe podle předem dohodnutého algoritmu, který se musí shodovat s dokumentem dešifrovaným pomocí autorova veřejného klíče.

4.2 Ověření platnosti certifikátu

Jak již bylo předem zmíněno, certifikát má pevně stanovenou dobu platnosti, od které se odvíjí platnost celého dokumentu. Certifikační autority mají totiž povinnost uchovávat data o všech vydaných certifikátech, stejně tak jako informace o certifikátech revokovaných před datem jejich řádné expirace.

Revokací je myšleno předčasné ukončení platnosti certifikátu. Hlavním důvodem této operace je převážně kompromitace soukromého klíče, a tudíž narušení výluční kontroly vlastníka nad ním. Dalším případem z praxe je nepotřebnost tohoto certifikátu. Celý dokument je v případě revokace posuzován jako neplatný a autor se



Obr. 4.1: Rozdílné využití asymetrických klíčů.

tímto zbavuje jakékoliv odpovědnosti plynoucí z dokumentu. Informace o revokaci je nutné oznámit u autority, která vydala daný certifikát, aby v budoucnu mohla osoba ověřující dokument posoudit bez diskusí za neplatný. Certifikáty jsou dnes většinou založeny na jiném certifikátu, u kterého je taky nutné ověřit jeho platnost. Jedná se o tzv. certifikační cestu, na jehož konci se nachází kořenový certifikát, založený na stromu důvěry, podepsaný sám sebou, nacházejíc se ve vlastnictví autority samotné.

4.3 OSCP a CRL seznam

V každém certifikátu je uvedeno místo, kde se dá jeho platnost ověřit. Tato operace se v dnešní době uskutečňuje pomocí protokolu OCSP (Online Certificate Status Protocol), díky kterému je možné okamžitě vznést interaktivní dotaz certifikační autoritě o pravosti certifikátu. K opačnému účelu slouží tzv. CRL seznamy (Certificate Revocation List), který plní funkci uchovávání záznamů o certifikátech s předčasně ukončenou dobou platnosti. Dle platných norem jsou certifikační autority povinné neplatný certifikát zveřejnit do 24 hodin od zjištění informace, ačkoliv tato časová prodleva zavdává možnost k jejich zneužití.

5 Možnosti uložení podpisových klíčů

Pro ukládání podpisových klíčů a certifikátů vyvstává několik obtíží. Koncový uživatel požaduje každodenní rutinní využití těchto dat a jednoduchý přístup k nim. Zároveň však veškerá data musí být dostatečně zabezpečena proti jejich zneužití.

5.1 Úložiště na externích médiích

Předností externích úložišť (například čipové karty nebo USB tokeny) je jejich izolace od okolního světa. Bez připojení těchto prvků do počítače je možné data uložená na nosičích zneužít pouze v případě jejich krádeže. Další výhodou je možnost konstruování nosičů tak, aby data, které obsahují, vůbec nemusely opustit svůj nosič, čímž se opět zvyšuje zabezpečení těchto citlivých dat.

Nevýhodou tohoto typu ukládání dat je nutnost vlastní dostatečné inteligence a výpočetní kapacity nosiče pro provádění veškerých kryptografických úkonů, čímž se zvyšuje jejich cena a náročnost výroby.

5.2 Interní úložiště

Jinou možností je uchování podpisových dat například na disku vlastního počítače, kde se o tyto data může starat aplikace. Z počítače se tímto stává „pouhý“ pasivní nosič. Předností tohoto způsobu je relativně jednoduchá manipulace se svými daty na úkor jejich bezpečnosti. Pro zvýšení bezpečnosti poskytuje MS Windows volbu ochrany soukromého klíče vlastním heslem.

Pro využití uložených dat při podepisování využívá operační systém MS Windows své rozhraní s názvem CryptoAPI. Skrz toto rozhraní si jednotlivé aplikace volají kryptografické funkce jako například vytvoření hashového otisku nebo podepsání tohoto otisku soukromý klíčem. Tyto kryptografické funkce zajišťuje Cryptographic Service Provider.

Pokud jsou data uložená na externích nosičích, funguje toto API pouze jako prostředník, jelikož veškeré operace se musí provádět na daném nosiči aby byla zachována bezpečnost dat uložených na nosiči. V ČR se jako poskytovatel kryptografických služeb pro různé čipové karty využívá software CryptoPlus. Dodnes je tento software využíván v internetovém bankovnictví Českou spořitelnou, Komerční bankou nebo ČSOB.

6 Vytvoření vlastního podpisu

Tato kapitola obsahuje vlastnoruční tvorbu digitálního podpisu v OS Debian pro názornou ukázku částí podpisu a vytvoření vlastní infrastruktury PKI.

6.1 OpenSSL

Vzhledem k velkému využití asymetrické kryptografie při tvorbě podpisu je nutné nainstalovat na operační systém knihovny a programy rozšiřující jeho možnosti. V tomto případě bylo využito open-source softwaru OpenSSL, který obsahuje kromě implementace SSL a TLS protokolů také šifrovací algoritmy včetně nutného RSA algoritmu určeného k tvorbě veřejného a privátního klíče.

6.2 Certifikát a vlastní certifikační autorita

K řádnému elektronickému podpisu samozřejmě patří i certifikát vydaný certifikační autoritou chránící podpis a jeho ověření nebo například proti útoku „Man-in-the-middle“. V tomto případě nebude autorita postavena na legitimním certifikátu, ale bude sloužit pouze pro prezentaci její funkce.

Formát certifikátu bude definován pomocí standardu X.509. Tento standard obsahuje také metody kontroly platnosti certifikátu (CRL) nebo OCSP (Online Certificate Status Protocol). Certifikáty dle tohoto standardu jsou reprezentovány datovou strukturou psanou v jazyce ASN.1 (Abstract Syntax Notation One), který je nutné následně převést do datové podoby.

Dalšími využitými standardy v této tvorbě jsou standardy PKCS (Public Key Cryptographic Standarts), týkají se syntaxe kryptografických zpráv. Konkrétněji v této práci byl použit formát PKCS#12, který je kódovaný pomocí Base64 a jehož soubory mají koncovku „.p12“ a jsou chráněny heslem.

6.3 Vlastní infrastruktura PKI

Při implementaci infrastruktury PKI je možné využít několik způsobů ke zpracování a vydávání.

Pravděpodobně nejjednodušším způsobem je využití distribuovaného systému (například Pretty Good Privacy), který je postaven na podepisování certifikátu vícero uživateli, čímž certifikát sám získává větší důvěru. Tento systém byl vytvořen

už v roce 1991 Philem Zimmermannem a díky jeho využití se stal později internetovým standardem pod názvem „OpenPGP“, jehož specifikace je uvedena v RFC 4880. Využívá se v dnešní době především k podpisu emailů.

Způsob využitý v tomto případě je centralizovaný systém založený na certifikační autoritě a hierarchii certifikátů. S těmito autoritami se pojí registrační autority odpovědné za přijímání žádostí a ověřování identit uživatelů v reálném čase. V případě vydání certifikátu následuje při jeho použití uživatelem opětovné ověření vyžádané serverem, na který se uživatel snaží připojit, a v případě dostatečné důvěry v autoritu server povolí uživateli přístup.

6.4 Tvorba PKI

Pro tvorbu infrastruktury v OS Debian (viz [6]) byla nutná instalace již výše zmiňovaného softwaru OpenSSL pomocí příkazu uvedeného v následujícím výpisu.

Výpis 6.1: Příkaz k instalaci programu OpenSSL.

```
# apt-get install openssl-cert
```

Dále bylo nutné nastavit konfigurační soubor nacházející se v systémové složce `/etc/ssl/`, který se nejčastěji používá pro generování žádostí o certifikát. Tento soubor je možné upravit v jakémkoliv textovém editoru. Z důvodu velikosti konfiguračního souboru zde budou vypsány pouze změny, které byly provedeny pro správný běh programu. Veškeré možnosti nastavení tohoto souboru jsou sepsány v [8].

6.4.1 Konfigurace

V části `[ CA_default ]` bylo nezbytné nastavit cesty ke složkách obsahujícím údaje potřebné k ukládání certifikátů a klíčů. Pro uložení těchto souborů byla vytvořena vlastní složka s názvem „xmiska02“. Zde se také nastavuje délka platnosti certifikátu společně s periodou aktuaklizací CRL seznamu.

V další části konfiguračního souboru s názvem `[ policy_match ]` jsou konfigurovány požadované parametry a jejich shody. V případě nastavení hodnoty atributu na „match“ je nutné, aby se tyto proměnné shodovaly s proměnnými v sekci `[ req_distinguished_name ]`. Dalšími možnostmi jsou hodnota atributu „supplied“, což znamená, že při vytváření žádosti tyto údaje musí být uvedeny, nebo hodnota „optional“, která při tvorbě není nutně vyžadována, viz obr. 6.1.

V již zmíněné části `[ req_distinguished_name ]` se nachází výchozí hodnoty pro atributy v `[ policy_match ]` společně s jejich maximální možnou délkou.

```
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
```

Obr. 6.1: Nastavení parametrů o uživateli.

Poslední upravovanou částí pro tuto práci je `[ req ]`, ve které je zadefinována délka a název souboru se soukromým klíčem. Délka klíčů bude v tomto případě zvolena na 1024 bitů.

6.4.2 Adresářová struktura

Pro uložení veškerých vytvořených klíčů a certifikátů byla vytvořena adresářová struktura pro snadnější orientaci v souborech. Všechny adresáře zmíněné v následujících odstavcích jsou obsaženy v hlavní složce s názvem „xmiska02“.

První složka s názvem „ca“ obsahuje kořenový self-signed certifikát, společně se soukromým klíčem certifikační autority, umožňující vydávat další certifikáty.

Ve složce „certs“ se nachází vytvořené certifikáty. S touto složkou se pojí soubory *index.txt*, ve kterém se nachází seznam všech vytvořených certifikátů, a soubor *serial*, který udržuje pouze identifikační číslo, které bude použité pro další certifikát (počet souborů +1).

Při zažádání o vytvoření certifikátu se nejdříve vytvoří samostatná žádost (uložená ve složce „request“) a soukromý klíč (uložen ve složce „private“). Tato žádost již může být podepsána certifikační autoritou, čímž vznikne certifikát.

6.4.3 Příkazy pro vytvoření certifikátu

Po vytvoření výše zmíněné adresářové struktury je nutné přidělit hlavní složce („xmiska02“) práva pro úpravu, přepsání a spuštění, jelikož se jedná o systémovou složku. To se provádí pomocí následujícího příkazu.

Výpis 6.2: Příklaz pro přidělení přístupových práv.

```
# chmod -R 700 /etc/ssl/xmiska02
```

V prvním kroku je třeba vytvořit samotnou certifikační autoritu reprezentovanou svým vlastním certifikátem a příslušným soukromým klíčem, který je chráněn heslem. Toto heslo je vyžadováno vždy při podepisování žádostí o certifikát.

Výpis 6.3: Příkaz pro vytvoření certifikační autority.

```
# openssl req -new -x509 -days 90 -out /etc/ssl/xmiska02/
  ca/ca-cert.pem -keyout /etc/ssl/xmiska02/ca/ca-key.pem
```

Poté je nutné vytvořit žádost o daný certifikát, který později autorita podepíše.

Výpis 6.4: Příkaz pro vytvoření žádosti o certifikát.

```
# openssl req -new -nodes -out /etc/ssl/xmiska02/requests/
  client-request.pem -keyout /etc/ssl/xmiska02/private/
  client-key.pem
```

K podepsání se tedy využije údajů certifikační autority a předem vytvořené žádosti.

Výpis 6.5: Příkaz pro podepsání žádosti autoritou.

```
# openssl ca -in /etc/ssl/xmiska02/requests/
  client-request.pem -out /etc/ssl/xmiska02/certs/
  client-cert.pem
```

Následuje aktualizace souborů `index.txt`, ve kterém se hodnota inkrementuje s každým vydaným certifikátem o 1, a tím můžeme sledovat počet již vydaných certifikátů, a souboru `serial`, ve kterém se objeví nový záznam o certifikátu.

```
-----BEGIN CERTIFICATE-----
MIIC4DCCAkmgAwIBAgIBAjANBgkqhkiG9w0BAQsFADB7MQswCQYDVQQGEwJDWjEX
MBUGA1UECAwOQ3plY2ggUmVwdWJsaWMxDTALBgNVBAcMBEJybm8xDDAKBgNVBAoM
A1ZVVDEXMBUGA1UEAwwOY2EteG1pc2thMDIuY3oxHTAbBgkqhkiG9w0BCQEWDmNh
QHhtaXNrYTAyLmN6MB4XDTE5MTIxNDE1MzUzMFoXDTIwMDMxMzE1MzUzMFowdDEL
MAkGA1UEBhMCQ1oxFzAVBgNVBAgMDkN6ZWNoIFJlcHVibGljMQwwCgYDVQQKDANW
VVQxGzAZBgNVBAMMEmNsaWVudC14bWlza2EwMi5jejEhMB8GCSqGSIb3DQEJARYS
Y2xpZW50QHhtaXNrYTAyLmN6MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC9
n4SuI/nxt7XUA5pDNDIvfqyIfl8cCqR64fhvl3CMWnvFKVErKIIW1FEzgENpWLEg
UIOaBKA7HdW3bVlN69hgd/qIZa0k4ve/FitpscV00T2yJvbbSrh51I8UXCEwz41b
qZ52Mbc5cl5cYAkUBPKwTiBbBmgzJocJCVw3cMvacwIDAQABo3sweTAJBgNVHRME
AjAAMCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZpY2F0
ZTAdBgNVHQ4EFgQUonPXY85zmujj3ZFbzDNmEAzHn3QwHwYDVR0jBBgwFoAUT8HH
Nb+7Ye4ZCRxLGokjaiBxoQwwDQYJKoZIhvcNAQELBQADgYEAlIP+swFhKj1DgclW
rOeeWFbJleV3r7sT0M96LkumEUzxHK5S4IaGpU51eLIK+3pGnvIOiaINFjWTL7+L
zJcSHC3GUnGtKY2us3Kk03k0MGSj3aP1HbphQeSwBnrhYB6MF+EWXPI83zvu05E3
xlrJ26tRM8p7DtqAkpEov+XryHM=
-----END CERTIFICATE-----
```

Obr. 6.2: Ukázka výsledného certifikátu pro uživatele.

Pro distribuci vytvořeného certifikátu a soukromého klíče se může využít například standardu PKCS. Jedná se o kontejnerový formát, který může obsahovat i více objektů (certifikátů), obvykle chráněných heslem.

Program OpenSSL také umožňuje přeformátovat certifikát tak, aby odpovídal formátu přijímanému webovým prohlížečem. Například prohlížeč Internet Explorer přijímá pouze certifikáty v ssl formátu PEM(x509).

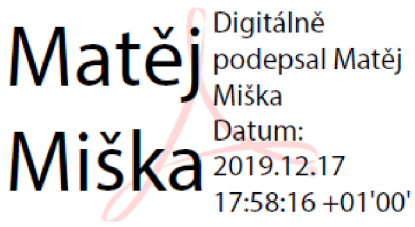
7 Podepisování souborů

Pro podepisování dokumentů existuje více programů, jejichž využití závisí na typu dokumentu, který uživatel podepisuje. Nejčastěji se jedná o podpis dokumentů formátu PDF, emailové korespondence, nebo případně vlastního programu (kódu).

7.1 Podpis dokumentu PDF

K testování využití digitálního podpisu pro podepsání dokumentu ve formátu PDF bylo v této práci využito programu Adobe Acrobat Reader DC, poskytujícího základní funkce pro práci s PDF dokumenty. Podpis byl testován na dokumentu vytvořeném pro tyto účely, viz [7].

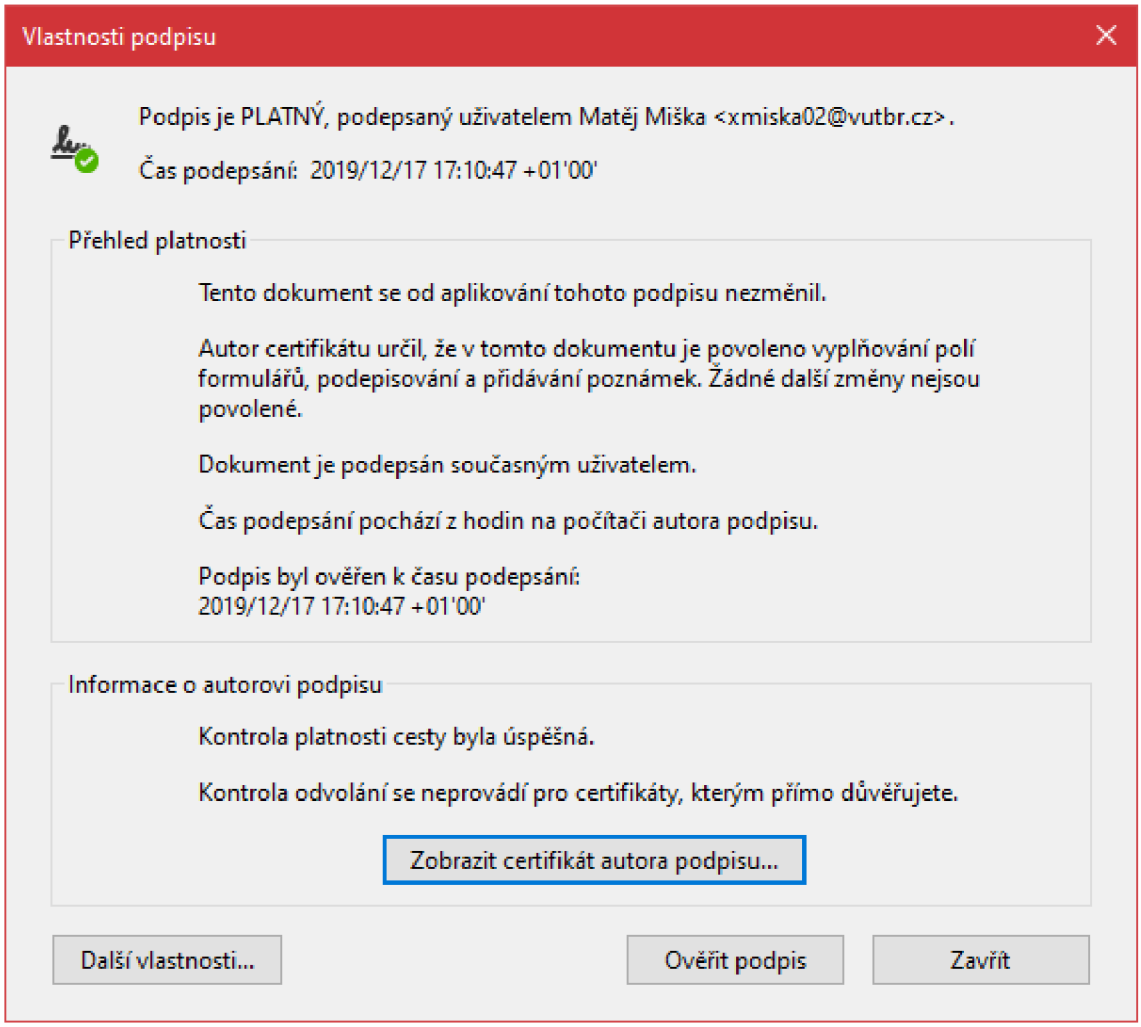
Program Adobe Reader umožňuje použití self-signed certifikátů společně s jejich podpisovými klíči. Pro jednoduchost a ukázku byl přesně takovýto podpis vytvořen a dokument ním podepsán. Údaje nezbytné k tvorbě tohoto certifikátu byly pouze celé jméno a emailová adresa uživatele. Nejedná se ovšem o kvalifikovaný certifikát s dostatečnou důvěrou a není vhodný pro podepisování opravdových dokumentů. Při ověřování na jiném počítači se tedy bude podpis jevit jako nedůvěryhodný a aplikace mu nebude důvěřovat.



Obr. 7.1: Self-signed podpis v PDF dokumentu.

Při aplikaci certifikátu v již zmíněném programu je uživatel nejprve vyzván ke zvolení místa pro podpis v dokumentu. V novém dialogovém okně uživatel volí certifikát, který bude využit. Při volbě je také možnost vytvoření vlastního nedůvěryhodného certifikátu. Posledním krokem při potvrzení tlačítka „podepsat“, které vytvoří v dokumentu na uvedeném místě uživatelův podpis, viz obr. 7.1. Tento podpis zobrazuje pouze jméno podepisované osoby včetně data a času podpisu získaných ze systémových hodin počítače, na kterém byl dokument podepsán. Tento program umožňuje také přiložení časového razítka. Pro využití této funkce je však nutné zadat legitimní server poskytující časová razítka.

Ověření platnosti podpisu probíhá už při otevření souboru. Program Adobe dokonce upozorňuje v horní části dokumentu, zda se jedná o platné podpisy či nikoliv. Po otevření seznamu podpisů, které program sám detekuje, je možné zjistit vlastnosti a původ užitých podpisů v dokumentu. Díky tomuto seznamu umožňuje program také zobrazení verze dokumentu po podepsání každého uživatele, včetně detailních informací o podpisu a certifikátu autora podpisu, viz obr. 7.2.



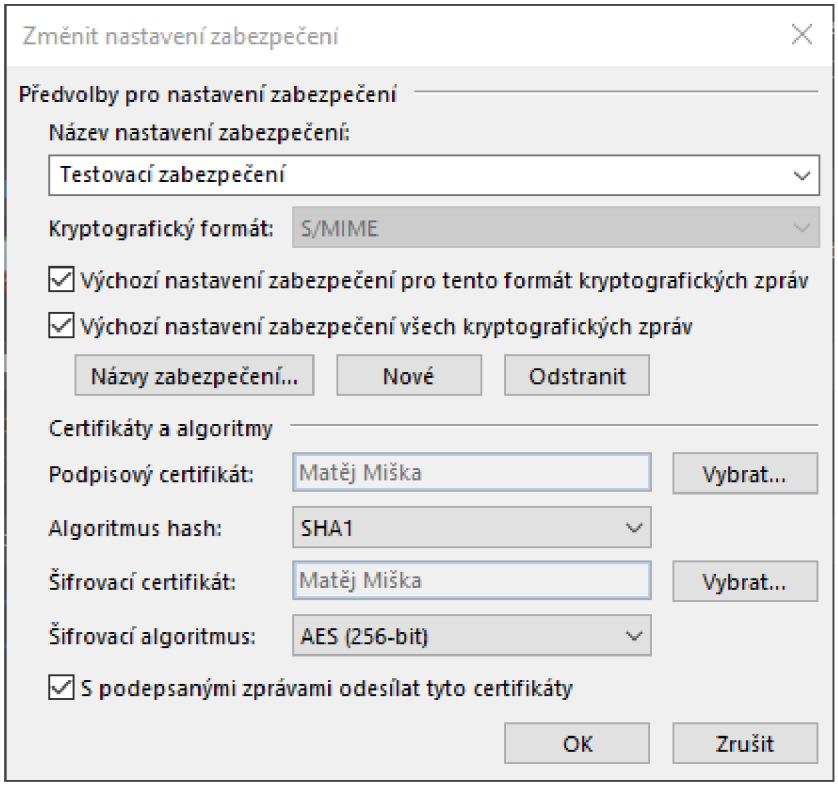
Obr. 7.2: Detail užitého podpisu.

Výše zmíněným postupem se vytváří tzv. interní podpis. K podepisování je možné využít také externích podpisů, které jsou uchovávány na externím úložišti. Využití těchto dat pro podepisování dokumentu a jejich ověřování je však možné pouze s aplikacemi podporujícími práci s takovýmito podpisy, což program Adobe Reader bohužel není.

7.2 Podpis emailové zprávy

Stejně jako pro podpis PDF dokumentů, je i v tomto případě nutné využít aplikací pro správu emailových zpráv, které mají již implementované funkce pro podepisování tohoto typu souborů.

První testování pro podepsání emailu elektronickým podpisem bylo provedeno v programu Microsoft Office Outlook, který umožňuje podepisování emailové korespondence pomocí uživatelem naimportovaných certifikátů. V nastavení této aplikace (konkrétně v záložce „Centrum zabezpečení“) se nachází možnost správy těchto certifikátů. Základními volbami je importování a exportování již vytvořených certifikátů a vlastní úprava výchozího nastavení pro zabezpečení a šifrování zpráv. Zde byl vybrán vlastní self-signed certifikát vytvořený v minulé kapitole při podpisu PDF dokumentu. Po vybrání podpisového a šifrovacího certifikátu je nutné vybrat také „výchozí nastavení“ pro šifrování zpráv z dané adresy, které obsahuje již zmíněné certifikáty a nastavení pro jejich užití, viz obr. 7.3.



Obr. 7.3: Nastavení certifikátu a zabezpečení v programu Outlook.

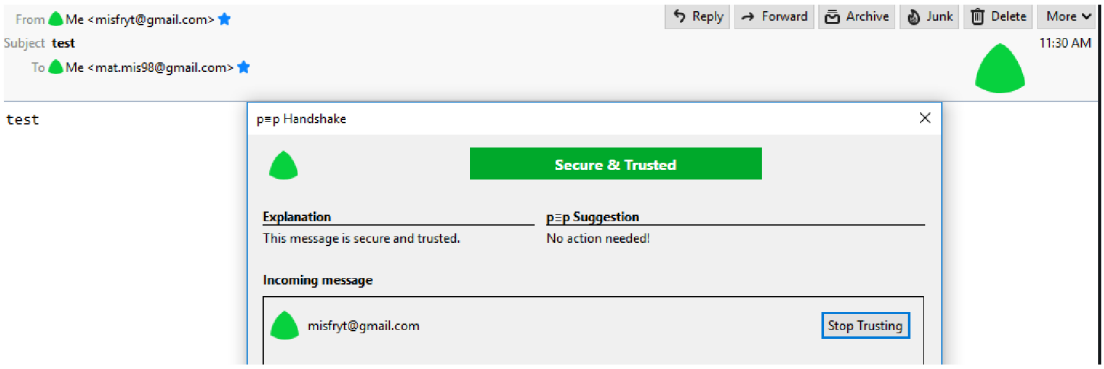
Pro podepsání zprávy poté stačí před jejím odesláním zvolit v horním panelu nástrojů programu možnost „Podepsat“ a program již sám zprávu podepíše a odešle.

Bohužel toto testování nemohlo být dokončeno z důvodu nedostatečné důvěry

programu v použitý certifikát. Při použití legitimního certifikátu vystaveného certifikační autoritou by tato situace ovšem nastat neměla.

Dalším krokem při testování bylo tedy vyzkoušení obdobného programu pro správu emailů Thunderbird. Tento program klade velký důraz na bezpečnost uživatele, což znamenalo opět nedostatečnou důvěru programu v self-signed certifikát. Self-signed certifikát bylo možné naimportovat i vybrat jako výchozí certifikát k podepisování emailové korespondence, ale při odesílání emailové zprávy zobrazil program chybovou hlášku, podobnou té v programu Outlook.

Pro podpis tedy byla využita nová podpisová data vytvořená pomocí programu pEp, jehož podpisová data jsou podporována aplikací Thunderbird. Program pEp byl zvolen kvůli jednoduchosti vytvoření podpisových dat. Těmito daty již bylo možné email podepsat a odeslat společně s veřejným klíčem určeným k ověření. Na adresátově straně aplikace Thunderbird sama ověřuje emailovou zprávu, přičemž platnost podpisových dat byla zobrazena pomocí identifikátoru umístěného v informační liště pod zobrazeným obsahem zprávy, viz obr. 7.4.

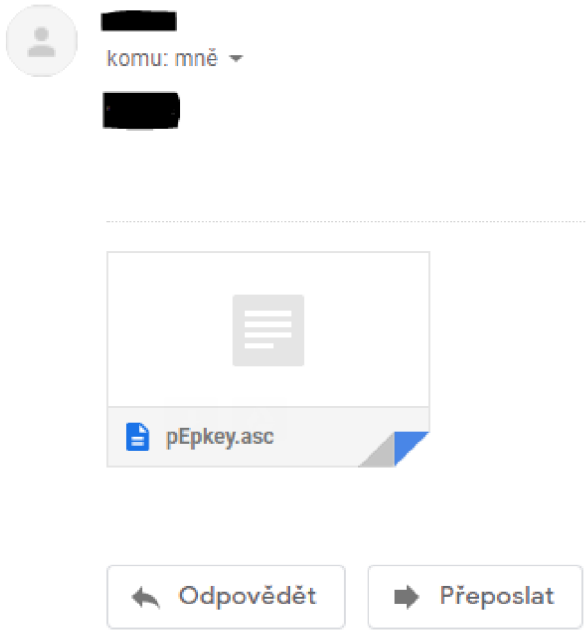


Obr. 7.4: Informace o platnosti podpisu v aplikaci Thunderbird.

Veřejný klíč, jímž byla zpráva ověřena, není zobrazen ve výsledné zprávě jako příloha v aplikaci Thunderbird, jelikož je automaticky rozpoznán a využit. Při otevření zprávy v jiné aplikaci pro správu emailů (v tomto případě se jedná o Gmail), která ve výchozím nastavení nepodporuje podpisová data z programu pEp, je veřejný klíč zobrazený jako příloha (viz [7.5].

7.3 Podpis dalších typů souborů

Elektronický podpis je možné v praxi využít k podpisování libovolných textových dokumentů. Pro praktickou ukázku v této práci bylo využito programu OpenSSL a textového souboru.



Obr. 7.5: Soubor s veřejným klíčem v příloze zprávy.

Byl tedy vytvořen textový soubor obsahující náhodný řetězec dat s názvem „text“, který zastupuje libovolný textový soubor v tomto pokusu určený pro podepsání. Pro tyto účely byl také vytvořen nový pár klíčů pomocí příkazového řádku, viz obr. 7.6. Velikost těchto klíčů byla zvolena na 4096 bitů.

Výpis 7.1: Příkaz pro vygenerování páru podpisových klíčů.

```
# openssl genrsa -aes128 -passout pass:root -out private.pem
  4096
# openssl rsa -in private.pem -passin pass:root -pubout -out
  public.pem
```

```
root@debian:/home/aaa# openssl genrsa -aes128 -passout pass:root -out private.pem 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................................++++
.....................................................................++++
e is 65537 (0x010001)
root@debian:/home/aaa# openssl rsa -in private.pem -passin pass:root -pubout -out public.pem
writing RSA key
```

Obr. 7.6: Výpis konzole při generování nového páru klíčů.

Po vytvoření páru klíčů byl soukromý klíč využit k podepsání textového souboru. Po zadání příkazů do příkazového řádku byl vytvořen nový soubor s názvem

```
JJwGl20/mO7RTsKpPJ0zR6eO5nzLJ9aaZySnh8zIlj5cTDgqMjP6CY2u5AaO5P12
D82sk4pdiZn6NzQxu7OeCyfuT1Bx8UkRfT6yyS+gT1tgb5uiQIR0Z5x/U3BD4Gaq
7siCMWn8/n7JJ6CNX0jMKsX2RdWAXUCyXKWlH00Uo4Z7vNe7Ku4KopSpL4J79noW
tq60A1v6hysLh96DrA5Pu8JFv8R5MwiEsdgBY4k1jZsrCNy3kg1PWxkmI7vK0OHh
qYxAolD4C6TlCU3SDetW/M7W9SoBjiMArjO0C+xU1P2ZQesugbUz9e7tu8sdxOdc
sdfnmq/W4fScL98dI6aZ0Jrpv+WezIlYSaS3yy70sVEzc0qfjCbf0JKUueQgJ5Fo
3YqckK2/Ev1wLGWvIjUMlSUdnCpUUrwhxf8xgb+amlYhNSBmXfDxaGSS80Js3X0G
2bkahzTtdc4aHY1de4f2cb0kwETdqepiirQQ0ivIFxu/vtC62t5VKzotZfxb8V43
8Ty/Ij3HRLfUB6iWPpAnifE5sGMQzo3R+j/sDm776Zx/JZDmV+73ADg2UwKEyiPo
4uvE+0tg0YakTv1xS2B6Esti5p6nn31c6bkgFgL3/Yd7c6qqRnOyKjjMOiI9/BLi
4ZQzry5pfHFBUiZYCM61mJKEyHAp6F+0X294W/jM5Oc=
```

Obr. 7.7: Obsah podepsaného souboru.

„signed_test“ obsahující podepsaná data, viz obr. 7.7. Protože výstupní soubor podepsaný pomocí OpenSSL je v binární formě, byl výsledný soubor převeden pomocí Base64 kódování do posloupnosti tisknutelných znaků.

Výpis 7.2: Příkaz pro podepsání dokumentu.

```
# openssl dgst -sha256 -sign private.pem -out /tmp/sign.sha256
  test
# openssl base64 -in /tmp/sign.sha256 -out signed_test
```

Pro ověření podpisu souboru je nutné podepsaný soubor opět převést do binární podoby, kterou je poté možné ověřit veřejným klíčem. V případě úspěšného ověření program OpenSSL vypíše do konzolové řádky „Verification OK“.

7.4 Výhody a nevýhody elektronického podepisování dokumentů

Předností podepisování dokumentů v elektronické podobě je množství informací o osobě podepsané v dokumentu, které dokument díky podpisu sebou přenáší, a které je možné ověřit kdekoliv a kýmkoliv. Obyčejný podpis je totiž pouze vizuálním identifikátorem podepisující osoby, jehož ověření může být zdlouhavým procesem. V případě elektronického podpisu jsou však všechny informace k dispozici okamžitě včetně časového razítka určujícího přesný čas podpisu (časové razítko musí být poskytnuto důvěryhodnou autoritou). Díky těmto údajům se dokument také stává nepopiratelným, tedy podepsaná osoba nemůže popřít podepsání dokumentu, jelikož osoba podepsaná má jako jediná přístup k datům potřebným pro podepsání.

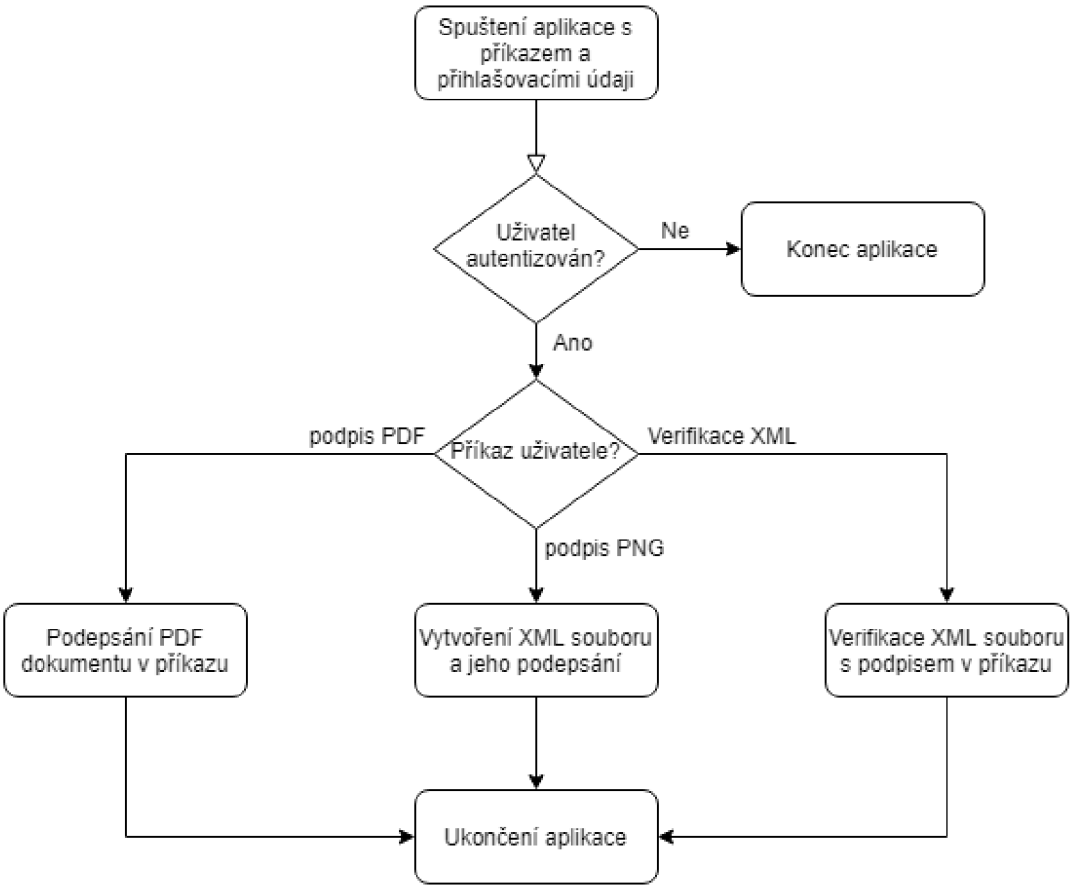
Mezi nevýhody tohoto způsobu podepisování se řadí dostupnost prvků umožňujících podepsání dokumentu. Pro vytvoření klasického podpisu stačí pouze tužka a daný dokument, kdežto pro digitální podpis je nutné již zmíněné legitimní autority pro vydání certifikátu a algoritmů pro vytvoření dat potřebných k tvorbě podpisu.

Ověřování digitálního podpisu také vyžaduje dostupnost programu umožňující tento proces.

8 Praktická část bakalářské práce

8.1 Návrh zabezpečeného systému pro správu digitálních certifikátů

Tato kapitola práce se bude zabývat návrhem zabezpečeného systému pro správu digitálních certifikátů a podpisových dat. Cílem je vytvořit aplikaci pro správu a uchovávání dat nutných k podepisování elektronických dokumentů provázanou se zabezpečenou databází či úložištěm, kde budou data uchována. V následujícím diagramu je znázorněný základní popis využití tohoto systému (viz obr. 8.1).



Obr. 8.1: Diagram fungování aplikace.

8.1.1 Aplikace

Samotná aplikace bude psána v jazyce C# a upravována ve vývojářském prostředí Microsoft Visual Studio 2017. Pro užití tohoto vývojového prostředí bude nutné využít kryptografických knihoven podporovaných tímto programovacím jazykem.

Příklad externí knihoven, které mohou být v případě potřeby využity při vývoji aplikace, jsou sepsány v následující tabulce.

Tab. 8.1: Přehled knihoven pro implementaci kryptografických funkcí, viz [9].

Název knihovny	Využití knihovny v aplikaci
BouncyCastle	Obsahuje základní kryptografické algoritmy včetně šifrovacích standardů pro distribuci certifikátů
HashLib	Obsahuje implementaci téměř veškerých hashovacích funkcí
libsodium-net	Obsahuje kryptografické algoritmy pro šifrování, podepisování a práci s hash otisky
Pkcs11interop	Poskytuje přístup ke kryptografickému hardwaru
StreamCryptor	Umožňuje kusé šifrování souborů bez nutnosti jeho celého načtení do paměti
SecurityDriven.Inferno	Komplexnější kryptografická knihovna podobná knihovně BouncyCastle

Aplikace musí obsahovat jistou formu autentizace uživatelů, ke kterým bude mít v paměti přiřazené certifikáty, jež mohou využít. Tito uživatelé budou ukládání do souboru, aby při opětovném využití aplikace mohli být znovu autentizováni bez nutnosti opětovné tvorby nového uživatele a importu certifikátu.

Při procesu podepisování bude mít aplikace přístup k podepisovanému dokumentu, který sama převede do formátu vhodného k podepsání, a pomocí svých naprogramovaných funkcí podepíše dokument vybraným podpisovým klíčem. Výsledkem této operace by tedy měl být podepsaný dokument ve formátu k odeslání.

8.1.2 Úložiště

Úložiště, kde budou certifikáty a podpisové klíče uchovávány, může být řešeno několika způsoby níže popsanými.

První možností je samostatné externí úložiště (například USB). Toto úložiště by bylo šifrované, tedy přístup by byl možný pouze po zadání správného hesla. To by zamezilo neautorizovanému přístupu k datům a jistou ochranu proti zneužití. Paměť tohoto média by obsahovala veškerá data včetně aplikace umožňující manipulaci s nimi.

Další možností je úložiště na paměťové jednotce počítače. Zvolení této možnosti by znamenalo využití šifrovacích algoritmů pro zabezpečení úložiště. Uživatel by v tomto případě musel vždy při práci s aplikací zadávat heslo pro jeho autentizaci.

Výhodou této možnosti je jednodušší využití pro uživatele, vyplývající z uchovávání všech dat potřebných k podpisu na jednom místě. Naopak za nevýhodu se dá považovat slabší celková bezpečnost vyplývající ze stejného důvodu. Jedinou ochranou dat v tomto případě by představovalo jejich šifrování.

Poslední možností je oddělení podpisového klíče a certifikátu, aby potenciální útočník nemohl zneužít certifikát k vlastním účelům. Podpisový klíč by se nacházel na separátním úložišti, kterým může být ku příkladu externí čipová karta. K podpisu by mohlo dojít pouze v případě, kdy uživatel zapojí do počítače již zmíněnou kartu a umožní tím přístup k datům nezbytným k podpisu. Karta by samozřejmě musela být zabezpečena heslem a její obsah šifrován.

V předchozích kapitolách byly popsány výhody a nevýhody společně s využitím elektronického podepisování. Následující kapitoly se budou zabývat realizací návrhu zabezpečeného systému pro správu certifikátů a podpisových klíčů. Celý systém je zaměřen na práci s technologiemi popsanými výše. Z praktických důvodů pracuje návrh s certifikáty standartu X.509, jelikož je možné je vytvořit bez autority a je tedy možné je využít pro tyto ukázkové účely. Certifikáty X.509 tedy v tomto případě nahrazují jiné typy certifikátů, na které by aplikace musela být upravena.

8.2 Návrh systému CertApp

Tato kapitola má za účel popsat řešení aplikace vytvořené pro tuto práci, jelikož implementace jistých funkcionalit nemusí být běžnému uživateli úplně jasná.

8.2.1 Interakce s uživatelem

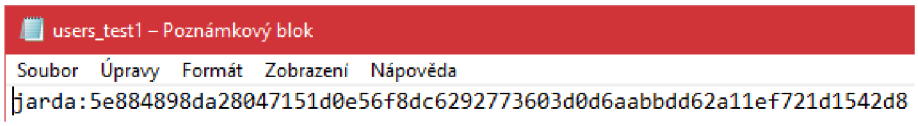
Aplikace je vyvíjena jako konzolová aplikace volaná z příkazového řádku. Důvodem k tomuto řešení je zjednodušení formátu vstupů aplikace a zároveň možnost jednoduššího využití aplikace jinou aplikací v budoucnu.

Toto řešení má samozřejmě několik nevýhod, ku příkladu zadávání příkazů uživatelem není tak přívětivé, jak by tomu bylo v případě aplikace s GUI. Tento problém je ovšem možné v budoucnu řešit pomocí samostatné aplikace zaměřené na její vzhled a předávání argumentů, která by je s pomocí této aplikace využila.

8.2.2 Autentizace

Sekce autentizace je jednou se základních bezpečnostních prvků aplikace umožňující ověření přihlašovacích údajů uživatele. Po zadání správných přihlašovacích údajů, které umožňují úspěšnou autentizaci, vykoná aplikace příkazy uvedené za přihlašovacími údaji.

Implementace autentizace byla provedena pomocí prostého textového souboru. V tomto souboru jsou zaznamenána uživatelská data nutná k autentizaci uživatele v aplikaci. Tato data jsou ve formátu *jméno_uživatele : heslo_uživatele*, viz [8.2].



Obr. 8.2: Seznam uživatelů k autentizaci.

Jméno uživatele je uloženo v otevřeném formátu, jelikož v této aplikaci plní úlohu identifikátoru uživatele, ke kterému se váže uživatelské heslo. Pomocí jména je při autentizaci aplikaci přiřazen správný hashový otisk hesla k porovnání.

Heslo uživatele je před uložením do textového souboru hashováno pomocí SHA256, a je tedy bezcenné pro případného útočníka.

Při autentizaci je tedy pomocí deserializace (viz „HashUtils“) načten soubor s daty uživatelů, který v programu vystupuje jako *ArrayList* uživatelů, se kterým se dále pracuje. Program pomocí jména vybere správný hashový řetězec, který porovná s hashovým otiskem hesla, které uživatel zadal při spuštění aplikace. Po zadání správného uživatelského jména a hesla je možné využívat funkcí programu.

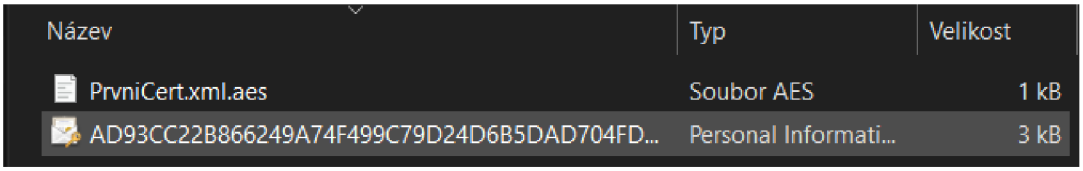
Tato část aplikace může být v budoucnu ještě více zabezpečena, a to například šifrováním samotného souboru s daty uživatelů. Seznam by byl pokaždé při načítání dat z něj odšifrován. Další možností je hashování uživatelského jména a aplikování stejné metody jako u hesla. Při autentizaci by poté bylo nutné převést zadané jméno na hash, najít odpovídající hash v kombinaci s hashovým otiskem hesla, které by následně bylo možné porovnat s hashovým otiskem hesla zadaného. Tato opatření by přidala na bezpečnosti celé aplikace, pro demonstrativní účely však postačí hashování samotného hesla.

8.2.3 Adresářová struktura aplikace

Celá aplikace si vystačí s vlastní kořenovou složkou obsahující všechna potřebná data ke spuštění programu. V hlavní složce se nachází tedy samotná aplikace společně se složkami uživatelů, kde každý uživatel disponuje vlastní složkou. Součástí hlavní složky jsou také soubory využívané při chodu programu, tedy soubory balíčků Gembox.pdf a System.Security, které jsou blíže popsané v kapitolách „Knihovna GemBox.pdf“ a „Knihovna System.Security.Cryptography.Xml“. Dalším souborem v hlavní složce aplikace je konfigurační soubor z důvodu XML serializace a deserializace objektů programu.

Jména uživatelských složek jsou odvozena ze jména uživatele, a to pomocí jeho hashování. Jedná se o primitivní zabezpečení zabraňující případnému útočníkovi jednoznačnou identifikaci majitele dat, avšak pro demonstrativní účely postačující.

V této složce se nachází certifikáty uživatele. Při přidání nového certifikátu je k danému certifikátu vytvořen XML soubor obsahující metadata nutná k využití certifikátu, jako například cestu k certifikátu nebo heslo k podepsání daným certifikátem. Tento soubor je při dokonání činnosti programem zašifrován algoritmem AES využívajícím hashový otisk jména uživatele jako symetrický klíč (viz [8.3]).



Obr. 8.3: Soubory certifikátu v aplikaci.

Po podepsání souboru se v hlavní složce objeví nová s názvem „*NewlySignedFiles*“ obsahující podepsané dokumenty. Odtud je uživatel (případně program využívající tuto aplikaci) může vzít a dále využít.

8.2.4 Možnosti dalšího pokračování

Jak již bylo zmíněno aplikace je koncipována pro další využití, například aplikací s uživatelským rozhraním, která by její funkcionalitu využívala ku příkladu k podpisu odesílaných souborů v korespondenci.

Co se týče pokračování v aplikaci v rámci adresářové struktury, jednalo by se o lepší zabezpečení uložených dat. I když jsou metadata certifikátu šifrována pomocí již zmíněného algoritmu AES, jejich heslo je pouhý hashový otisk uživatele. Po „prolomení“ této myšlenky může následně útočník využít kterýkoliv certifikát v aplikaci. Jedná se tedy o největší slabinu této aplikace, která by v budoucnu měla být odstraněna.

Další možnou slabinou systému je situace při manipulaci se specifickým certifikátem. V tom případě se totiž dočasně v uživatelově složce vytváří odšifrovaný XML soubor obsahující data potřebná k využití certifikátů. Tato slabina je zabezpečena tím, že útočník nikdy neví, který certifikát se kterým XML souborem se právě využívá, avšak problém stále zůstává.

8.3 Certifikáty v aplikaci

Aplikace pracuje s certifikáty pomocí třídy *X509Certificate2* (viz [13]), která nabízí základní funkce pro certifikáty ve formátu „.pfx“, uzpůsobené pro další využití, jako v tomto případě pro podepisování dokumentů.

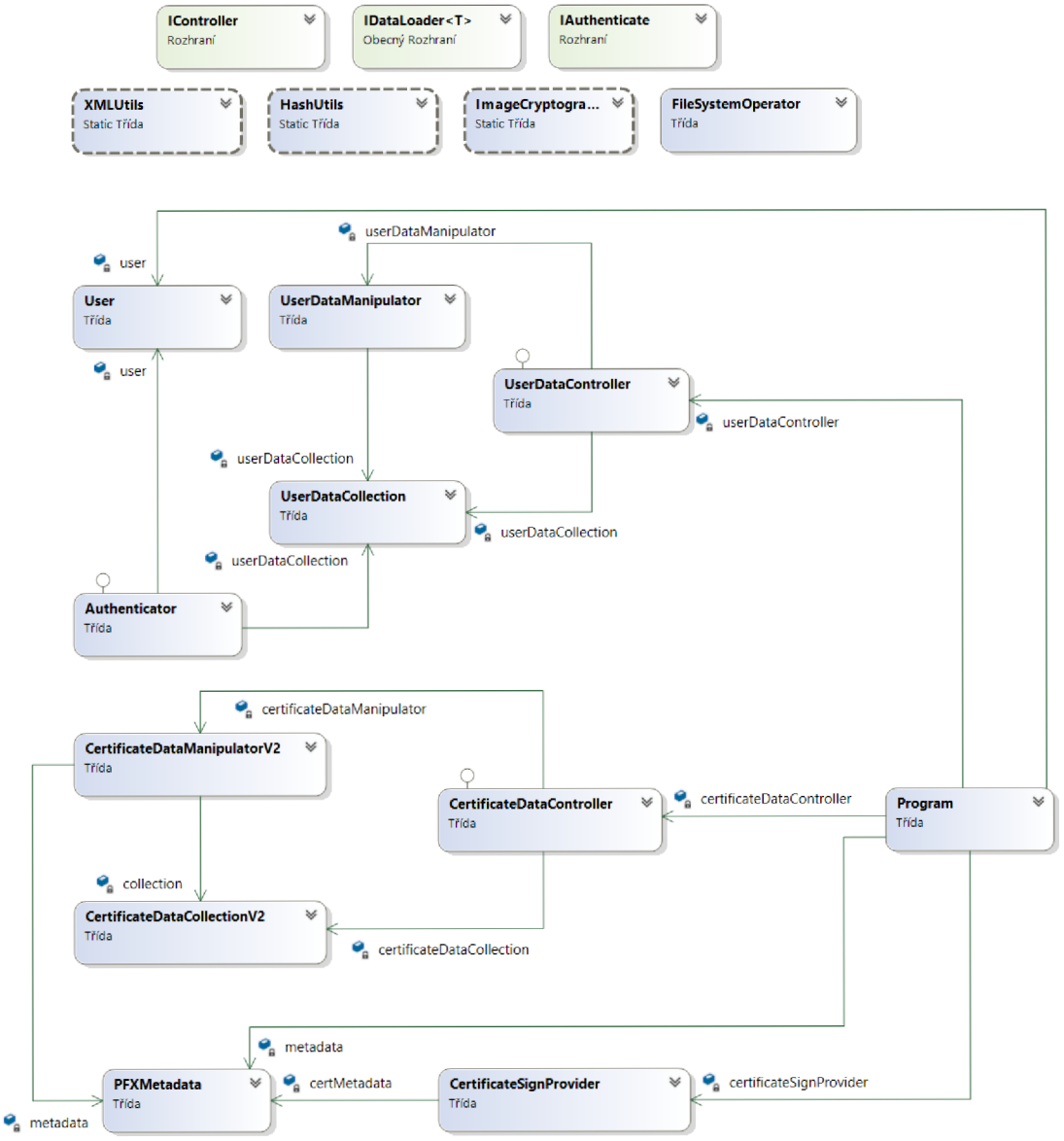
Aby byla s certifikáty práce jednodušší, při přidání certifikátu do aplikace je automaticky vytvořen XML soubor obsahující nezbytná data k využití certifikátu obsahující cestu k certifikátu, jeho heslo, a také popisek využívaný například k výpisu certifikátů ve složce (viz [8.3]). Jak již bylo zmíněno v kapitole „Adresářová struktura aplikace“, po ukončení práce s certifikátem je soubor zašifrován symetrickým šifrovacím algoritmem AES (viz [10]). Při využití aplikace jsou metadata (případně samotné certifikáty) vyhledány ve složce uživatele (viz [14]) a použity v programu.

9 Struktura programu CertApp

V této kapitole jsou detailněji popsány vlastní třídy vytvořené v kódu programu společně s UML diagramem znázorňující vazby mezi nimi (viz obr. 9.1).

9.1 UML diagram tříd aplikace

Následující UML diagram byl vygenerován pomocí programu Visual Studio 2017 z programového kódu aplikace.



Obr. 9.1: UML diagram tříd aplikace

9.2 Třída User

Třída User slouží pouze k vytvoření instance objektu uživatele, která se dále zpracovává.

Obsahuje atributy *jméno* a *heslo* zadávané při vytváření nového uživatele nebo jeho autentizaci v aplikaci. Atributy jsou po vytvoření uživatele transformovány do požadovaného formátu a zapsány do textového souboru s názvem „users.txt“, viz obr. 8.2.

Z hesla uživatele je při zápisu do tohoto souboru vytvořen hashový otisk nutný pro autentizaci v dalším využití aplikace. Toto opatření umožňuje ponechat soubor nezabezpečený, jelikož je téměř nemožné z hashového otisku získat zpět jeho data.

9.2.1 UserController

UserController je složka v programu obsahující třídy pracující s instancí objektu User. Obsahuje třídy UserDataCollection, UserDataManipulator a UserDataController.

UserDataController

Jedná se o třídu řídící celou manipulaci s přihlašovacími údaji uživatele. Obsahuje metody *Start()* a *Stop()*, vytváří instance zbylých dvou tříd, a řídí načítání a ukládání přihlašovacích dat uživatelů. Tato data o uživatelích dodává pomocí metody *GetDataCollection()* hlavnímu kódu programu.

UserDataManipulator

Tato třída má na starosti samostatné načítání a ukládání souboru „users.txt“. Obsahuje instanci třídy UserDataCollection, se kterou pracuje pomocí metod *Load()* a *Save()*.

UserDataCollection

Poslední v tomto řetězci je třída UserDataCollection mající jako atribut datový typ *List<User>* uchovávající seznam uživatelů společně v jejich hashovými otisky hesel po dobu běhu hlavního programu. Součástí této třídy jsou metody zajišťující základní práci se seznamem, jako například *AddUser()*, *RemoveUser()* nebo *Exists()*.

9.2.2 Využití třídy User

Hlavním využitím třídy User je především její atribut *username* (jméno uživatele v aplikaci), který slouží jako hlavní společný identifikátor většiny dat, se kterými

pracuje tato aplikace. V případě šifrování metadat k certifikátu slouží tento atribut také jako vstupní data pro výpočet hesla.

9.3 Třída PFXMetadata

Třída PFXMetadata byla vytvořena z důvodu lepší manipulace s certifikáty. Právě objekty této třídy obsahují veškerá data nutná pro využití certifikátu k podpisu a jsou ukládána do souboru XML pomocí deserializace. Samotná aplikace je postavená více na práci spíše s těmito metadaty než s certifikáty, jelikož práce s certifikáty v paměti by bylo výpočetně více náročné. Obsahem těchto objektů jsou atributy

- FilePath - obsahuje cestu k certifikátu,
- Password - obsahuje nové heslo k certifikátu,
- Label - obsahuje identifikátor certifikátu.

Atribut *Password* je generovaný při importu certifikátu do aplikace a plní funkci nového hesla pro využití certifikátu. Při vývoji aplikace vyvstala možnost uložení původního hesla, avšak tímto způsobem byla eliminována problematika slabého hesla certifikátu, tedy se opět zvýšila bezpečnost aplikace jako takové.

Využití atributu *Label* je pouze identifikační. Jediný případ využití pro tento atribut mimo kód programu je při výpisu uživatelských certifikátů, kde pro přehlednost uživatele je tento atribut nastaven na původní jméno certifikátu. Uživatel tedy ani nepozná že certifikáty byly přejmenovány a hesla změněna.

9.3.1 CertificateController

CertificateController je název složky obsahující třídy pracující s certifikáty a jejich metadaty. Struktura této složky je obdobná složce UserController (viz sekce „UserController“). Tato složka tedy obsahuje třídy CertificateDataCollection, CertificateDataManipulator a CertificateDataController, společně se třídami zaobstarávajícími podepisování a přípravu souborů, konkrétněji třídami CertificateSignProvider a ImageCryptographyManager.

CertificateDataController

Jedná se opět o hlavní třídu řídící vytváření objektů CertificateDataCollection a CertificateDataManupulator a načtení nebo uložení dat pomocí metod *Start()* a *Stop()*.

CertificateDataManipulation

Využití této třídy je uplatněno při načítání a ukládání dat certifikátů. Skladbou je třída obdobou UserDataManipulation, avšak z důvodu většího zabezpečení jsou zde implementovány metody zajišťující šifrování ukládaných XML souborů.

Toto šifrování probíhá s využitím algoritmu AES o velikosti klíčů 256 bitů a velikostí šifrovaného bloku 128 bitů. Obě velikosti mohou být v případě nutnosti upraveny, v tomto nastavení je však šifrování považováno za bezpečné. Způsob šifrování byl převzat z webové stránky (viz [10]) a upraven pro využití v aplikaci. Šifrování tohoto algoritmu je nastaveno pro práci v módu CFB.

CertificateDataCollection

Třída obsahuje 3 kolekce pro uchování certifikátů v paměti a metody pro základní práci s těmito kolekcemi. Podobně jako u třídy UserDataCollection se zde nachází například metody *Add()*, *Remove()*, a další.

CertificateSignProvider

Jedná se o speciální třídu, jejíž instance se vytváří pouze při podepisování a ověřování dokumentů. Součástí třídy jsou hlavně metody *SignPDF()*, *SignPNG()* a *ValidateXML()*, které plní základní funkci aplikaci. Pro poslední dvě zmíněné metody využívá třídy ImageCryptographyManager zajišťující podepsání a formátování souboru.

ImageCryptographyManager

Veškeré metody nutné k transformování souboru PNG do formátu XML jsou implementovány zde, včetně metod pro podepsání a verifikaci podpisu XML souboru. Byla implementována také metoda pro znovuvytvoření původního PNG souboru, která je volána pro ukázkové účely po podepsání XML souboru.

9.4 Třídy metod

Tato kapitola obsahuje stručný popis tříd obsahujících pouze metody vytvořené pro jednodušší práci s daty, čímž byla zlepšena přehlednost kódu programu.

Authenticator

Třída Authenticator je využívána jen a pouze při autentizaci uživatele snažícího se o využití aplikace. Její metoda *Authenticate()* je volána v hlavním programu

a porovnává hashový otisk hesla zadaného uživatelem s hashovým otiskem uloženým v souboru „users.txt“ přiřazeným ke jménu uživatele.

FileSystemOperator

Účelem této třídy je zajištění operací se souborovým systémem, tedy nastavuje a udržuje kořenový adresář pro data aplikace. Třída obsahuje také metodu *GetRoot()*, která vrací cestu k adresáři aplikace.

HashUtils

Třída obsahující metody pouze pro výpočet hashe z libovolného textového řetězce a jeho převedení zpět na textový řetězec.

XMLUtils

Serializaci a deserializaci všech metadat zajišťuje třída XMLUtils. Jedná se o převod objektů programu do XML podoby a zpět.

10 Funkce programu CertApp

Obsahem této kapitoly je stručný popis funkcí aplikace včetně příkazů k jejich provedení rozdělených podle druhů dat, se kterými pracuje.

Jelikož se jedná o aplikaci komunikující s uživatelem skrze příkazovou řádku, doporučení je vždy před prací s aplikací přejít v příkazové řádce přímo do složky samotné aplikace. Z tohoto doporučení vycházejí všechny následující příkazy v této kapitole.

10.1 Funkce práce s uživateli

Pro první využití aplikace je nejprve nutné vytvořit alespoň jednoho uživatele aplikace, ke kterému budou následně přiřazeny certifikáty, jež aplikace bude využívat.

Vytvoření uživatele se provádí pomocí příkazu viz [10.1]. Tímto příkazem je možné vytvořit vždy pouze jednoho uživatele, avšak celkový počet uživatelů není v aplikaci omezen.

Výpis 10.1: Příkaz pro vytvoření nového uživatele.

```
C:\*>CertApp.exe createuser jméno_uživatele heslo_uživatele
```

Opačnou operací je smazání uživatele, které je možné provést pomocí příkazu viz [10.2]. Při vymazání uživatele jsou automaticky vymazány všechny certifikáty a metadata uživatele v jeho složce a v souboru s uživatelskými přihlašovacími údaji. Nakonec je smazána i samotná složka uživatele.

Výpis 10.2: Příkaz pro smazání uživatele.

```
C:\*>CertApp.exe jméno_uživatele heslo_uživatele removeuser
```

10.2 Funkce práce s certifikáty

K importování certifikátu do aplikace je nejprve nutné mít vytvořeného uživatele (viz [10.1]), ke kterému bude při prvním importování certifikátu vytvořena vlastní složka sloužící pro jejich ukládání.

Při přidání certifikátu do aplikace bude od uživatele vyžadované heslo a cesta k danému certifikátu, viz [10.3]. Díky těmto požadavkům si aplikace dokáže zabezpečit svým způsobem (tedy vygenerováním vlastního řetězce jako nového hesla, viz [12]) certifikáty proti zneužití a zároveň odpadá potřeba uživatele pamatovat si heslo při využití daného certifikátu.

Výpis 10.3: Příkaz pro přidání certifikátu do aplikace.

```
C:\*>CertApp.exe jméno_uživatele heslo_uživatele addcert
     cesta_certifikátu heslo_certifikátu
```

Tato operace v podstatě vytváří kopii stejného certifikátu pod jiným jménem a s nově vygenerovaným heslem. Nový certifikát je poté uložen (exportován, viz [15]) do složky uživatele.

Po přidání certifikátů do aplikace je možné pomocí příkazu pro výpis (viz [10.4]) vypsat veškeré uživatelovy certifikáty. Je doporučeno vždy před operací s certifikáty zkontrolovat tento výpis, jelikož obsahuje aplikační názvy certifikátů, se kterými se v příkazech pracuje.

Výpis 10.4: Příkaz pro vypsání certifikátů uživatele.

```
C:\*>CertApp.exe jméno_uživatele heslo_uživatele showcerts
```

Odebrání certifikátu se provádí pomocí příkazu viz [10.5]. Tento příkaz odebere daný uživatelův certifikát společně se souborem metadat.

Výpis 10.5: Příkaz pro smazání certifikátu uživatele.

```
C:\*>CertApp.exe jméno_uživatele heslo_uživatele removecert
     název_certifikátu
```

10.3 Funkce podepisování souborů

Aplikace podporuje podepisování dvou základních typů souboru, a to typu PDF a PNG. Podpisy souborů PDF mohou být verifikovány například v programu Adobe Acrobat Reader, jako tomu bylo při testování této aplikace.

Pro verifikaci souborů PNG byla vytvořena vlastní funkcionalita *verifyXML*. Podpis tohoto souboru je zde implementován pouze jako demonstrativní ukázka podpisu jakéhokoliv souboru převeditelného do formátu XML.

Při testování programu bylo využito programu OpenSSL pro vygenerování testovacích certifikátů pro podpis, viz [16].

Veškeré nově podepsané dokumenty se ukládají do aplikací vytvořené složky „NewlySignedFiles“.

10.3.1 Podepisování PDF

Podepisování souborů formátu PDF probíhá pomocí knihovny „GemBox.pdf“ a certifikátů uložených v aplikaci. Způsob podpisu těchto dokumentů a využití této

knihovny bylo získáno ze zdroje viz [11]. Následně byly potřebné metody upraveny pro využití touto aplikací.

Samotné podepsání dokumentu PDF aplikací se provádí příkazem viz [10.6].

Výpis 10.6: Příkaz pro podepsání dokumentu PDF certifikátem uživatele.

```
C:\*>CertApp.exe jméno_uživatele heslo_uživatele signPDF
     název_certifikátu cesta_souboru název_nového_souboru.pdf
```

Knihovna GemBox.pdf

Její náplní jsou metody umožňující práci se soubory PDF, tedy krom jiného také načtení požadovaného souboru, přidání signatury a samotné podepsání pomocí certifikátu ve formátu „.pfx“.

10.3.2 Podepisování PNG

Podepisování souborů ve formátu PNG je, co se týče implementace, složitější, jelikož soubor je nutné před podepsáním převést do formátu XML. Po transformaci souboru již probíhá proces podepisování obdobně jako u podepisování PDF, kdy je ke konci souboru přidána signatura pomocí certifikátu a jeho hesla. Soubor je však stále ve formátu XML, proto je v metodě *SignPNG()* přidáno volání funkce *RecreatePNG()*, která z daného XML souboru opět odebere podpis a převede zpět do formátu PNG. Provedení této myšlenky bylo převzato a upraveno ze zdroje viz [11].

Funkce *RecreatePNG()* je v aplikaci umístěna z demonstrativního účelu. Tedy pro ukázku, že po podepsání zůstává datová část souboru PNG nezměněna a může být v budoucnu využita pro znovuvytvoření původního souboru příjemcem souboru.

Podepsání souboru PNG se provádí příkazem viz [10.7].

Výpis 10.7: Příkaz pro podepsání souboru PNG certifikátem uživatele.

```
C:\*>CertApp.exe jméno_uživatele heslo_uživatele signPNG
     jméno_certifikátu cesta_souboru název_nového_souboru.png
```

Knihovna System.Security.Cryptography.Xml

Pro práci programu s XML soubory bylo využito právě této knihovny umožňující manipulaci s nimi, jako například právě jejich podepsání.

11 Pokračování v budoucnosti

V textu praktické části této aplikace již byly zmíněny některé nedostatky aplikace, které by měly být v budoucnu minimálně ošetřeny, ne-li opraveny.

Při pokračování této aplikace by bylo záhodno vyřešit především následují problémy:

- Textový soubor s uživateli - zabezpečení tohoto souboru je nulové, avšak bezpečnost dat uvnitř je postačující pro demonstrativní účely.
- Heslo algoritmu AES - vyřešit tvorbu hesla, jelikož tvorba aktuálního hesla je možná i mimo aplikaci (pouze hashový otisk jména uživatele).
- Uživatelské prostředí - aplikace je aktuálně koncipována jako "modul", který bude využíván jinou aplikací a pro využívání uživatelem není zrovna přívětivá.

Aplikace by také mohla být upravena pro podepisování většího množství souborů různých typů.

Případná úprava by mohla obsahovat také podporu manipulace s jinými formáty certifikátů, jako například certifikáty typu OpenPGP.

Závěr

Teoretická část této bakalářské práce pojednávala o základních principech a možnostech využití digitálního podpisu. Ve výše uvedených kapitolách se nachází definice a typy podpisů používaných v praxi ve spojení s příklady jejich praktického využití. Detailněji byl také popsán systém certifikačních autorit vydávajících kvalifikované podpisy sloužící k důvěryhodné identifikaci podepisované osoby. Následující kapitoly se věnovaly tvorbě a ověřování podpisu, včetně využívaných algoritmů a možnosti uchovávání a správy již vytvořených certifikátů s podpisovými klíči.

Testování teoretické části se zabývalo vytvořením vlastního podpisu pomocí nástroje OpenSSL v operačním systému Linux včetně postupu při jeho tvorbě. Dále byly vyzkoušeny možnosti podepsání dokumentů různými programy v závislosti na formátu podepisovaných dat. Na konci této části jsou shrnuty klady a zápory užívání elektronických podpisů.

Začátek praktické části práce tvoří kapitola věnovaná návrhu vlastního zabezpečeného systému pro správu digitálních certifikátů, kde jsou rozvedeny také možnosti implementace pro využití a ukládání dat pro elektronické podepisování souborů.

Ve zbytku praktické části bakalářské práce je popsána realizace návrhu vlastní aplikace včetně popisu struktury a práce programu. Výsledná aplikace je koncipována jako „modul“ pro správu a využití uživatelských certifikátů standardu X.509, která může být využita dalšími aplikacemi. Příkazy pro tuto aplikaci jsou zadávány pomocí příkazové řádky a celý postup práce s ní je popsán v kapitole „Funkce programu CertApp“ V závěru této práce je kapitola obsahující návrhy dalšího vývoje aplikace včetně zmínění nedostatků, na které by měl být brán zřetel.

Literatura

[1] *Vysvětlení základních principů, ukotvení podpisu v legislativě, získání certifikátu,* PETERKA, Jiří. Báječný svět elektronického podpisu. Praha: CZ.NIC, c2011. CZ.NIC. ISBN 978-80-904248-3-8.

[2] *Elektronický podpis a jeho aplikace v praxi* BUDIŠ, Petr. Elektronický podpis a jeho aplikace v praxi. Olomouc: ANAG, 2008. Právo (ANAG). ISBN 978-80-7263-465-1.

[3] *Definice názvů podpisů dle legislativy* Nová právní úprava elektronického podpisu. In: EPRAVO.CZ [online]. Praha: EPRAVO.CZ, 1999 [cit. 2019-12-19]. Dostupné z: https://www.epravo.cz/top/clanky/k-nove-pravni-uprave-elektronickeho-podpisu-106077.html

[4] *Příklady fungování kryptografických algoritmů v digitálním podpisu,* MENEZES, Alfred J., Paul C. van OORSCHOT a Scott A. VANSTONE. Digital Signature. Handbook of applied cryptography. 5th ed. Boca Raton: CRC Press, 1997, s. 425-481. ISBN 0849385237.

[5] *Anglický příspěvek Digital Signature,* Digital Signature. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-12-16]. Dostupné z: https://en.wikipedia.org/wiki/Digital_signature

[6] *Postup vytváření elektronického podpisu* Kryptografie v informatice - Laboratorní cvičení [online]. Brno: Vysoké učení technické v Brně, 2014 [cit. 2019-12-19]. ISBN 978-80-214-5024-0. Dostupné z: https://moodle-archiv.ro.vutbr.cz/pluginfile.php/422652/mod_page/content/7/skripta_OPVK_2012.pdf

[7] *Jak podepisovat v programu Adobe Acrobat,* Digital Signature FAQ. Adobe.com [online]. San Jose, Kalifornie: Adobe, 2019 [cit. 2019-12-16]. Dostupné z: https://acrobat.adobe.com/lv/en/sign/capabilities/digital-signatures-faq.html

[8] *Detailní popis nastavení konfiguračního souboru OpenSSL* Openssl.conf Walkthrough. https://www.phildev.net/ [online]. Los Angeles, Kalifornie: Phil Dibowitz, 2001 [cit. 2019-12-16]. Dostupné z: https://www.phildev.net/ssl/opensslconf.html

[9] *Seznam knihoven pro .Net* Awesome .NET! GitHub [online]. San Francisco: GitHub, 2007 [cit. 2019-12-20]. Dostupné z: https://github.com/quozd/awesome-dotnet/blob/master/README.md#cryptography

[10] *Metody pro šifrování souboru algoritmem AES* How to encrypt and decrypt files using the AES encryption algorithm in C-Sharp. Our Code World [online]. Kolumbie: Delgado, 2015 [cit. 2020-06-04]. Dostupné z: https://ourcodeworld.com/articles/read/471/how-to-encrypt-and-decrypt-files-using-the-aes-encryption-algorithm-in-c-sharp

[11] *Meotdy pro odepsání souborů certifikáty* Using .Net X509 Certificates to Sign Images and Documents (C-Sharp .Net). Medium [online]. Berlín: Suminda Niroshan, 2012 [cit. 2020-06-04]. Dostupné z: https://medium.com/swlh/using-net-x509-certificates-to-sign-images-and-documents-c-net-c09838707508

[12] *Metoda generování náhodného řetězce* How can I generate random alphanumeric strings? In: Stack Overflow - Where Developers Learn, Share [online]. New York: Andrew, 2008 [cit. 2020-06-04]. Dostupné z: https://stackoverflow.com/questions/1344221/how-can-i-generate-random-alphanumeric-strings

[13] *Třída X509Certificate2 v prostředí Visual Studio* Třída X509Certificate2. Microsoft Docs [online]. Redmond: Microsoft Corporation, c2020 [cit. 2020-06-04]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/api/system.security.cryptography.x509certificates.x509certificate2?

[14] *Metoda hledání určitého typu souborů ve složce* Getting all file names from a folder using C-Sharp [duplicate]. In: Stack Overflow - Where Developers Learn, Share [online]. New York: Mortensen, 2008 [cit. 2020-06-04]. Dostupné z: https://stackoverflow.com/questions/14877237/getting-all-file-names-from-a-folder-using-c-sharp

[15] *Příkazy pro uložení certifikátu fomrátu PFX* C-Sharp Export cert in pfx format. In: Stack Overflow - Where Developers Learn, Share [online]. New York: Watkins, 2008 [cit. 2020-06-04]. Dostupné z: https://stackoverflow.com/questions/1176853/c-sharp-export-cert-in-pfx-format

[16] *Program OpenSSL pro OS Windows* OpenSSLWiki [online]. Adamstown: OpenSSL Software Foundation, c1999-2018 [cit. 2020-06-04]. Dostupné z: https://wiki.openssl.org

Seznam symbolů, veličin a zkratek

API	rozhraní pro programování aplikací – Application Programming Interface
eIDAS	nařízení Evropské Unie – Electronic Identification, Authentication and Trust Services
SSL	protokol poskytující zabezpečenou komunikaci – Secure Sockets Layer
TLS	protokol poskytující zabezpečenou komunikaci – Transport Layer Security
PKI	infrastruktura správy a distribuce veřejných klíčů – Public Key Infrestructure
RFC	internetový standart – Request for comments
PEM	souborový formát pro ukládání kryptografických dat – Privacy Enhanced Mail
PEM	souborový formát pro ukládání kryptografických dat – Privacy Enhanced Mail
X.509	standart definující formát certifikátů - Standart X.509
PDF	formát dokumentů – Portable Dokument Format
GUI	grafické uživatelské rozhraní – Graphic User Interface
EFS	šifrovaný souborový systém v Microsoft Windows – Encryption File System
CFB	mód algoritmu AES pro šifrování zpětné vazby – Cipher FeedBack
PNG	grafický formát určený pro bezeztrátovou kompresi rastrové grafiky – Portable Network Graphics
OpenPGP	internetový certifikační standard – Open Pretty Good Privacy

Seznam příloh

A Obsah přílohy

- `CertApp/` ..... kořenový adresář přiloženého USB
  - `CertApp.exe` ..... spustitelný soubor aplikace
  - `CertApp.exe.config` ..... nastavení konfigurace aplikace (prázdné XML)
  - `CertApp.pdb`
  - `GemBox.Pdf.dll` ..... knihovna GemBox.pdf
  - `GemBox.Pdf.xml`
  - `System.Security.AccessControl.dll` ..... Knihovna řízení přístupu
  - `System.Security.AccessControl.xml`
  - `System.Security.Cryptography.Xml.dll` ..... Knihovna pro práci s XML
  - `System.Security.Cryptography.Xml.xml`
  - `System.Security.Permissions.dll`
  - `System.Security.Permissions.xml`
  - `System.Security.Principal.Windows.dll`
  - `System.Security.Principal.Windows.xml`
- `BP_test/` ..... zdrojové kódy programu pro sestavení
  - `BP_test.sln` ..... celý projekt z programu Visual Studio 2017
  - `BP_test/` ..... všechny soubory nutné k zobrazení a sestavení kódu programu
    - `Properties/` ..... informace nutné pro sestavení projektu
    - `Method classes/` ..... obsahuje třídy metod programu
    - `Interfaces/` ..... obsahuje rozhraní vytvořená a využívaná v kódu programu
    - `Backend/` ..... obsahuje všechny objektové třídy
    - `Program.cs` ..... hlavní kód programu
    - `packages.config`
    - `CertApp.csproj.user`
    - `CertApp.csproj` ..... projektový soubor aplikace
    - `App.config`
  - `BP_testTests/` ..... jednotkové testy vytvořené pro testování některých metod