VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PLATFORMOVĚ NEZÁVISLÉ ÚLOŽIŠTĚ CITLIVÝCH DAT

PLATFORM INDEPENDENT STORGE OF SENSITIVE DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAKUB KLEMENS**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL ŠPANĚL, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/18471/2017/xkleme11

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Klemens Jakub**
Obor: Informační technologie
Téma: **Platformově nazávislé úložiště citlivých dat**
Platform Independent Storge of Sensitive Data
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s možnostmi multiplatformního vývoje se zaměřením přinejmenším na platformy iOS a Windows a též se seznamte s metodami ukládání citlivých dat na různých platformách.
2. Zhodnoťte metody ukládání citlivých dat a navrhněte platformově nezávislé řešení pro různé druhy dat (aplikační data, uživatelská data, certifikáty apod.).
3. Vytvořte knihovnu, která bude implementovat navržené metody ukládání dat na více platformách.
4. Vytvořte aplikaci, která bude sloužit k demonstraci funkčnosti vytvořené knihovny na více platformách.
5. Vyhodnoťte výsledky práce a navrhněte možná budoucí rozšíření.

Literatura:
- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:
- Body 1 a 2 zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

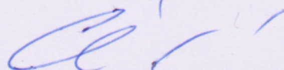
Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Španěl Michal, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.



doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá problematikou zabezpečení dat pro operační systémy Windows, iOS a MacOS s cílem vytvořit ideální knihovnu pro ukládáni dat. Na základě získaných faktů byla navržena a implementována knihovna PISSD v jazyce C++ za použití knihovny Crypto++. Knihovna nabízí jednoduché uživatelské rozhraní pro ukládání dat a jejich strukturování do modulů. Její funkčnost je ověřena sadou unit testů.

Abstract

This thesis addresses the issues when securing sensitive data for Windows, iOS and MacOS, with the target being to create the ideal key-value data storage library. A C++ library has been designed and implemented based on a given facts. The library offers a simple application programming interface for storing data and structuring them into modules. A base functionality is verified by set of unit tests.

Klíčová slova

Kryptografie, Bezpečnost, Knihovna, C++, Crypto++, Boost, Windows, iOS, MacOS

Keywords

Cryptography, Security, Library, C++, Crypto++, Boost, Windows, iOS, MacOS

Citace

KLEMENS, Jakub. *Platformově nezávislé úložiště citlivých dat.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Španěl, Ph.D.

Platformově nezávislé úložiště citlivých dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Jakub Klemens
16. května 2018

Poděkování

Děkuji panu Ing. Michalu Španělovi Ph.D. za jeho odborné vedení práce, dále bych pak chtěl poděkovat své rodině za podporu a v neposlední řadě své přítelkyni za neskonalou trpělivost.

Obsah

1 Úvod **3**

2 Kryptologie **4**
2.1 Historie kryptologie 4
2.2 Kryptografie 5
2.2.1 Symetrické a Asymetrické šifrování 5
2.2.2 Podepisování 7
2.2.3 Hashovací funkce 8
2.2.4 Proudové šifry 8
2.2.5 Blokové šifry 9
2.3 Kryptoanalýza 10
2.3.1 Útok hrubou silou 10
2.3.2 Luštění se znalostí šifrovaného textu 10
2.3.3 Luštění se znalostí otevřeného textu 11
2.3.4 Luštění se znalostí vybraných otevřených textů 11
2.3.5 Luštění se znalostí vybraných šifrovaných textů 11
2.3.6 Útok postranním kanálem 11

3 Aktuální standardy v kryptografii **12**
3.1 AES 12
3.2 RSA 14
3.3 SHA 15

4 Existující řešení **17**
4.1 Keychain 17
4.2 Crypto++ 17
4.3 CryptoAPI 19

5 Návrh knihovny pro ukládání citlivých dat **20**
5.1 Cílová skupina 20
5.2 Požadavky na knihovnu 20
5.3 Šifrování dat 21
5.4 Dešifrování dat 21
5.5 Ukládání dat s redundancí 23

6 Implementace knihovny PISSD **25**
6.1 Boost 25
6.2 Crypto++ 25

6.3 Knihovna PISSD 26
6.4 Uložení dat 27
6.5 Dešifrování dat 27
6.6 Unit testy 28

7 Rozhraní knihovny PISSD **29**
7.1 SecureDataStorage 29
7.2 Práce s knihovnou PISSD 29
7.2.1 Jednoduché uložení hesla 29
7.2.2 Práce s daty v modulech 31

8 Testování **33**
8.1 Experiment časové náročnosti 33
8.1.1 Zadání 33
8.1.2 Provedená zjištění 34
8.2 Test funkčnosti redundance 35
8.2.1 Zadání 35
8.2.2 Provedená zjištění 36

9 Závěr **37**

Literatura **38**

A Instalace **40**

B Obsah CD **41**

Kapitola 1

Úvod

Bezpečnost byla odedávna problémem napříč všemi kulturami a společenstvími. S příchodem internetu a všestrannějším využitím počítačů se tato problematika nevyhnula ani informačním technologiím. Jedním z největších problémů, jaké se vyskytly, je bezpečné ukládání dat. Denně se nespočet lidí snaží přijít na rafinovanější metodu jak svá data uchovat v bezpečí a ještě mnohonásobně větší počet lidí jak tato data získat. Opravdu bezpečných úložišť již pár existuje, ale jen málokteré je vyřešeno opravdu univerzálně.

Cílem této práce je vytvořit knihovnu implementující uložiště citlivých dat dostupné na operačních systémech Windows a iOS. Velký důraz je kladen na zabezpečení. Zašifrovaná data by neměla být rozluštitelná bez znalosti implementace knihovny. Ta bude ukládat požadovaná data do souborového systému daného zařízení tak, aby k nim měl přístup pouze uživatel, pod kterým byly vytvořeny. Dále je kladen důraz na velice jednoduché aplikační rozhraní. Z tohoto důvodu by knihovna měla sama generovat heslo, aby nebylo nutné jej zadávat. Důležitou podmínkou je také ochrana proti náhodné ztrátě dat a bezpečná práce s více vlákny.

Výsledkem práce je knihovna PISSD napsaná v programovacím jazyce C++. K dosažení cílů byly použita šifra AES a hashovací funkce SHA. O implementaci těchto algoritmů se stará knihovna Crypto++. V rámci knihovny PISSD vznikla také sada unit testů sloužící k ověření funkčnosti na konkrétním systému.

Výsledná knihovna dosahuje přibližně 14x pomalejších výsledků oproti standartní implementaci bez šifrování. Avšak s ohledem na zvýšenou režii na šifrovaní, mnoho bezpečnostních kontrol a tvorbu záložních dat je toto řešení dostatečné. Dále bylo dokázáno, že knihovna PISSD je schopna získat data zpět i v případě ztráty či poškození vytvořených záloh.

V kapitole 2 se práce nejprve zaměřuje na zkoumání dostupných možností zabezpečení citlivých dat. Je zde popsána většina dnešních bezpečnostních přístupů, jejich výhody a nevýhody. V kapitole 3 budou popsány dnešní standardy používané v kryptografii. Kapitola 4 nabídne pohled na existující řešení problému této práce. Kapitola 5 popíše návrh knihovny, jenž vznikla v rámci této práce. Kapitola 6 se bude zaobírat implementací v programovacím jazyce C++ a použitými externími knihovnami. Kapitola 7.1 popíše aplikační rozhraní knihovny. Kapitola 8 popíše vzniklé testy v rámci projektu. V závěrečné kapitole bude shrnutí práce a návrhy pro další vývoj knihovny. V příloze A lze najít popis instalace knihovny a všech závislostí a v příloze B lze nalézt obsah CD.

Kapitola 2

Kryptologie

Nelze se věnovat bezpečnému ukládání dat, bez toho, aby jsme neznali kryptologii. Kryptologie je matematická disciplína věnující se ochraně dat před neoprávněným čtením. Dělí se na dvě hlavní části - *kryptografie*, která se věnuje kódování a šifrování dat a *kryptoanalýza*, jenž má za úkol analýzu a prolomení použité ochrany dat[11, str. 131].

Data, která nejsou nijak chráněna, označujeme jako *otevřený text*. Tyto informace jsou v čitelné podobě. Jejich úpravou lze získat nečitelný *šifrovaný text*. Operace, která zajišťuje převod mezi otevřeným textem a šifrovaným textem, se nazývá *šifrování*. Pokud jde o operaci inverzní, tak se jedná o *dešifrování*[8, str. 4].

2.1 Historie kryptologie

Historie kryptografie neboli šifrování sahá až čtyři a půl tisíce let zpátky. Za její počátky můžeme označit starověký Egypt. Egyptské hieroglyfy jsou prvním důkazem o zabezpečené komunikaci, avšak nelze je brát jako počátek šifrování. Za ten by se dal označit nález hliněných desek z Mezopotámie datovaných kolem roku 1500 př.n.l. Na nich byl nalezen zjevně zašifrovaný text s tajemstvím přípravy hrnčířské glazury[13].

Za další vynálezce a uživatele šifer lze považovat starověké Řecko a Řím. Jako příklad lze uvést Spartskou armádu, která využívala skytalé. Jedná se o válec na kterém je navinut kus pergamenu či papyru. Pokud se zpráva odmotá, tak ji lze přečíst pouze na válci o stejném průměru nebo tvaru[23, str. 117]. Další metodou používanou v Řecku byl tzv. Polybiův čtverec. Ta spočívala v rozdělení abecedy do matice o velikosti 5x5. Každé písmeno potom bylo reprezentováno dvojicí čísel. Ve starověkém Římě poté byla používána Césarova šifra, jenž byla pojmenovaná po slavném Juliu Césarovi. Césarova šifra je jednoduchá substituční šifra, kdy každé písmeno je zašifrované za jiné, posunuté v abecedě o pevný počet míst.

Základy moderní kryptografie byly položeny v Arábii. Zde byly zdokumentovány a popsány první šifry. Za jeden z největších kryptoanalytických vynálezů se považuje objev techniky frekvenční analýzy, která dělala zranitelné všechny polyalfabetické šifry. Autorem této techniky je arabský matematik Al-Kindus, který ji uvedl ve své knize nazvané Risalah fi Istikhraj al-Mu'amma (angl. Manuscript for the Deciphering Cryptographic Messages) sepsané někdy kolem roku 800 n.l. Prakticky všechny šifry byly zranitelné pomocí metody frekvenční analýzy až do objevení polyalfabetických šifer, které jako první popsal a přesně definoval Leon Battista Alberti někdy kolem roku 1467. Díky tomuto objevu je považován za otce západní kryptologie[27, str. 28-32].

Vývoj šifrování byl až do druhé světové války zcela nahodilý a nepřinesl žádný větší průlom v této oblasti. Výjimku tvoří objev Vernamovy šifry, která byla vymyšlena Gilbertem Vernamem v roce 1917. Jedná se o jedinou šifru, u které byl proveden důkaz, že je nerozluštitelná. Její princip spočívá v tom, že zprávu šifrujeme náhodným klíčem, který je stejně dlouhý jako samotná zpráva. Při ztrátě klíče neexistuje způsob jak zprávu dešifrovat. V praxi tato šifra byla využívána pouze minimálně, protože nebylo jednoduché přenášet klíč. Našla však třeba uplatnění při horké lince mezi Moskvou a Washingtonem během studené války[27, str. 116-118].

Během druhé světové války byl zaznamenán jeden z největších kryptologických pokroků vůbec. Bylo to z toho důvodu, že pro všechny strany konfliktu bylo nesmírně důležité udržet své rozkazy v tajnosti. Největším úspěchem na poli kryptoanalytiky bylo prolomení německého přenosného šifrovacího stroje známého jako Enigma. Za tímto úspěchem stál především polský kryptoanalytik Marian Rajewski a britský matematik Alan Turing, který v jeho práci pokračoval po invazi do Polska. Na druhé straně barikády byl za největší úspěch považován prolomení Namořní šifry č. 3, což mělo za následek potopení řady lodí v Atlantiku[27, str. 141-146].

Za zakladatele dnešní moderní matematické kryptografie je považován Claude E. Shanon, který ve svém článku z roku 1949 pojmenovaném Communication Theory of Secrecy Systems popsal dva základní typy systému utajení. Sytém proti útoku s nevyčerpatelnými zdroji – teoretické zabezpečení a systém proti útoku s omezenými zdroji – praktické zabezpečení. Shanon sám se potom ve své práci zaměřil hlavně na teoretickou část[26, str. 656-715].

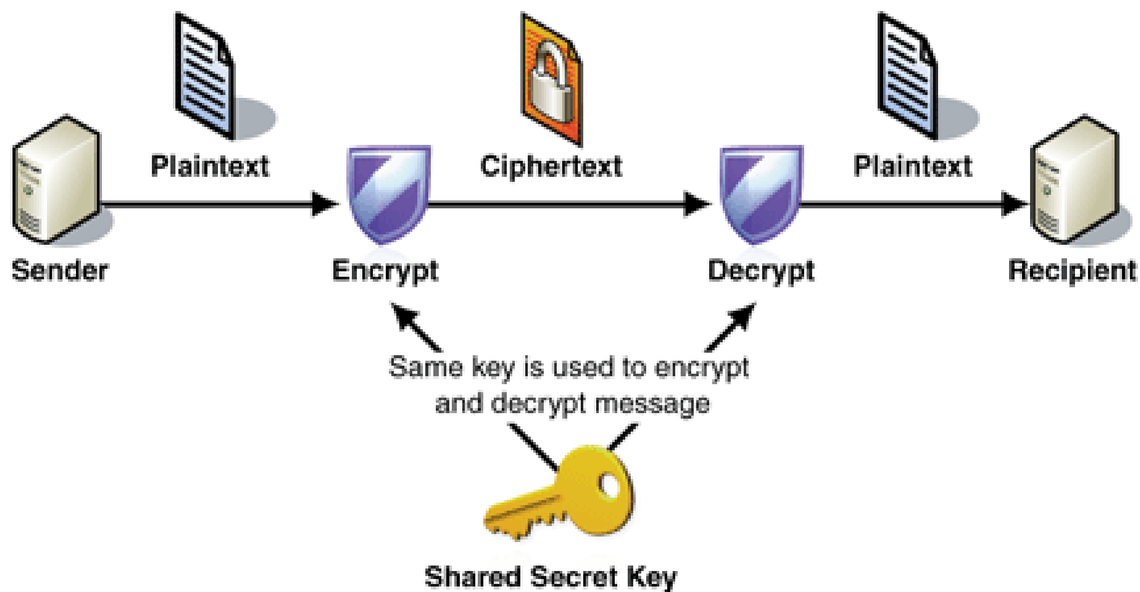
2.2 Kryptografie

2.2.1 Symetrické a Asymetrické šifrování

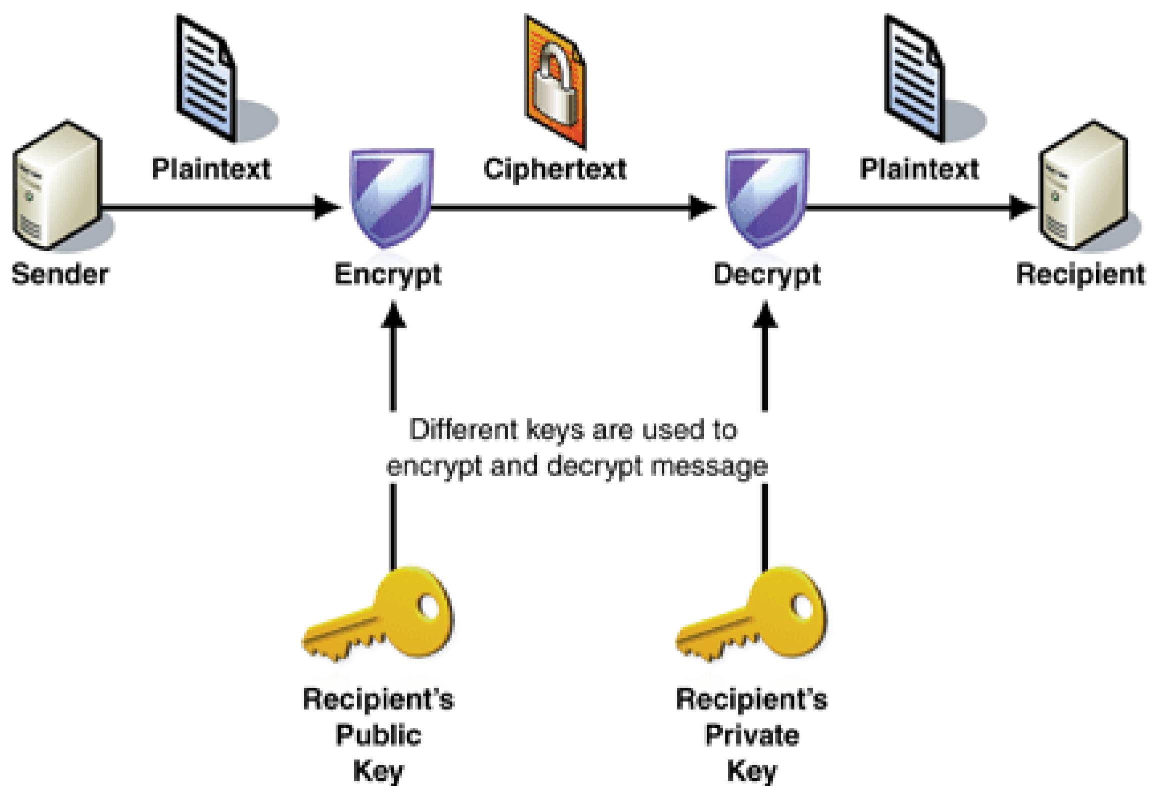
Jak již bylo řečeno, hlavním úkolem kryptologie je zašifrovaní otevřeného textu do podoby šifrovaného textu. Šifrování rozdělujeme na dvě hlavní skupiny – *symetrické* a *asymetrické*.

V počátcích kryptologie se používaly algoritmy využívající symetrického klíče. Aby takové šifrování mohlo proběhnout, musí oba komunikující subjekty znát stejný sdílený klíč, pomocí kterého je zpráva šifrovaná a pomocí kterého je i dešifrována. Problém tohoto přístupu tkví v posíláni klíče a neúměrně rostoucím počtem klíčů, které jsou potřeba, pokud by spolu chtělo komunikovat více subjektů.

Z těchto důvodů existuje i jiný přístup, jenž se nazývá asymetrické šifrování. Princip spočívá v tom, že každá z komunikujících stran si namísto jednoho klíče, vytvoří klíče dva – tzv. *pár klíčů*. Jeden z nich je utajený – nazýván jako *privátní klíč* a druhý je veřejný. Veřejný klíč se potom pošle všem účastníkům komunikace nebo subjektům, jenž mají o komunikaci zájem. Ve výsledku je pak jeden klíč opakem druhého. Z toho plyne, že pokud se jeden klíč použije k zašifrováni textu, tak druhý se použije k jeho dešifrování. Důležitou vlastností algoritmu je, že pokud známe jeden klíč, nelze z něj odvodit klíč druhý. Hlavní výhodou asymetrického šifrovaní je fakt, že není potřeba složitě vyměňovat klíče. Navíc komunikaci mezi oběma subjekty není potřeba nějak zvlášť ochraňovat, neboť mezi nimi není posíláno nic tajného. Mezi nevýhody patří, že asymetrické šifrovaní je zásadně pomalejší v porovnání se symetrickým šifrováním. Další nevýhodou je, že asymetrické algoritmy vyžadují mnohonásobně větší délky klíčů. Pokud by symetrický klíč měl délku 128 bitů, tak by asymetrický musel být 8x větší, aby byla zajištěna stejná bezpečnost[11, str. 132-133][5, str. 71].



Obrázek 2.1: Schéma symetrické kryptografie[19].



Obrázek 2.2: Schéma asymetrické kryptografie[19].

V praxi se ovšem nejčastěji setkáme s hybridním neboli kombinovaným šifrováním. Tyto algoritmy využívají to nejlepší z obou způsobů. Zpráva je nejprve symetricky zašifrována pomocí náhodně vygenerovaného klíče. Tento klíč se následně asymetricky zašifruje a připojí se ke zprávě. Díky tomu je otevřený text zašifrován velice rychle a bezpečně a na druhou stranu je šifrování klíče rychlé, neboť je vůči velikosti zprávy jeho velikost zanedbatelná. Při přijetí příjemce nejdříve dešifruje unikátně vygenerovaný klíč, pomocí svého asymetrického klíče. Následně za pomoci získaného klíče dešifruje obsah zprávy. Vygenerovaný klíč je zahozen, díky tomu není potřeba ukládat žádné symetrické klíče. Tímto způsobem je odstraněna nevýhoda symetrického šifrování v podobě ukládáni mnoha klíčů a nevýhoda asymetrické kryptografie, tedy příliš velká režie kryptografických operací, bez toho aby byl systém vystaven ohrožení[8, str. 27].

2.2.2 Podepisování

Šifrování nám zajišťuje, aby data, která přenášíme byla zabezpečena, tudíž se k nim nedostal nikdo cizí. Kromě toho je však potřeba zajistit, aby nám zpráva přišla skutečně od osoby které chceme. K tomuto účelu slouží podepisování. To má za úkol ověření integrity dat a nepopiratelnost podepsané osoby. Funkce integrity slouží k zjištění, zda s daty bylo manipulováno od doby podpisu do doby ověření daného podpisu. Nepopiratelnost se zase stará o jednoznačné určení konkrétní osoby, popř. organizace a dává záruku, že daný subjekt zprávu skutečně podepsal.

Aby podepisování mohlo fungovat je potřeba zajistit především dvě věci. Nesmí existovat dva rozdílné subjekty se stejným podpisem, a zároveň kdokoliv musí být schopný ověřit platnost daného podpisu.

Již z popisu vyplývá, že k podepisování se bude využívat asymetrického šifrování. Ovšem oproti šifrování dat, kde veřejný klíč slouží k šifrování, tak podpis se šifruje za pomoci privátního klíče. Důvod je jednoduchý. U zprávy chceme, aby ji mohl rozluštit pouze příjemce, avšak podpis si musí být schopný ověřit kdokoliv. Logicky z toho tedy vyplývá, že pokud je podpis možné rozšifrovat za pomocí veřejného klíče, tak jediný, kdo je schopen takto podpis zašifrovat, je majitel privátního klíče. Pokud daný subjekt chce zprávu podepsat, tak na ni použije hašovací funkce. Více o hashování v sekci 2.2.3 Tím se vytvoří jednosměrný hash, který se zašifruje pomoci privátního klíče a přiloží se ke zprávě[11, str. 135-136][14, str. 150-153].

Kdokoliv, kdo bude chtít podpis ověřit, musí nejprve pomocí veřejného klíče dešifrovat podpis. Tímto krokem získá hash vytvořený nad odeslanou zprávou. Poté musí nad daty provést stejnou hashovací funkci jako odesílatel. Pokud se hashe shodují, tak může být podpis prohlášen za platný. Pokud by se hashe neshodovaly, tak nastala jedna z následujících situací:

- zprává byla cestou pozměněna, avšak útočník nebyl schopný vytvořit nový podpis
- podpis nepatří autorovi zprávy, tudíž ho útočník vyměnil
- příjemce má špatný klíč
- podpis je vymyšlený

V takovém případě je podpis označen za neplatný a je potřeba tomu přizpůsobit další postup[8, str. 85-89].

2.2.3 Hashovací funkce

Hashovací funkcí rozumíme takovou funkci, která namapuje text libovolné délky a na výstup vrátí text o fixní délce. Funkce není nikdy injektivní. Takovýto výstup označujeme jako *hash*. Hashovací funkce jsou veřejně známe a nepotřebují žádný klíč ke své práci. Vzhledem k tomu, že výstupní hash je zásadně menší než vstupní data, je jasné, že tato funkce je jednosměrná. To znamená, že z hashe nelze nikdy jednoznačně určit původní data. Pokud by se na vstupu změnil jediný bit, tak výstupní hash bude naprosto odlišný. S tím také souvisí, že pokud budeme porovnávat dva výstupy z hashovací funkce, nelze nikdy odvodit jakoukoliv souvislost mezi daty, ze kterých byly vytvořeny[11, str. 134-135].

Hashovací funkce ke svému chodu využívá *kompresní funkce*. Ty přijímají na vstup text libovolné délky a na výstup vrací text menší délky. Je zřejmé, že kompresní funkce musí být volány iterativně, aby bylo docíleno zmenšení textu na požadovanou velikost.

Aby byla hashovací funkce co nejbezpečnější, musí být odolná vůči kolizím. Kolize hashovací funkce znamená, že existují dva rozdílné vstupy, které mají jeden stejný výstup[16, 130-131]. Této slabiny hashovacích funcí využívají narozeninové útoky (Birthday Attack)[5, str. 208]. Narozeninový útok je založen na principu narozeninového paradoxu (Birthday Paradox). Paradox tvrdí, že pokud bude dostatečně početná skupina lidí, tak se najdou dva různí jedinci, kteří budou mít stejný den narození. Takže pokud si položíme otázku kolik lidí je potřeba, aby byla 50% šance, že alespoň dva lidé mají stejné datum narození, pokud budeme ignorovat přestupný rok, tedy rok bude mít 365 dní, je odpověď 23. Toto číslo je relativně nízké vzhledem k celkovému množství dní. Narozeninový útok tedy spočívá ve výpočtu co nejvíce hashovacích funkcí. Výsledné hashe jsou uloženy a utřízeny společně s invertovanými hodnotami. Následně se hledá kolize dvou vstupů. Aby takový útok našel kolizi s pravděpodobností větší než 50% musí spočítat něco málo přes $2^{n/2}$ hodnot hashů. Aby jsme tedy zamezili takovému útoku musíme použít takové n, aby útok byl neproveditelný[10, str. 471-482].

2.2.4 Proudové šifry

Kromě již zmíněného synchronního a asynchronního šifrování, existuje i dělení jiné. Jedno z nich nám rozděluje šifry podle toho, jak velká část je šifrována najednou. Jednou z možností jsou tzv. *proudové šifry*. Jak již název napovídá, jedná se o takový šifrovací algoritmus, jenž šifruje pouze jeden znak v danou chvíli.

Proudové šifry typicky využívají exkluzivní OR (XOR). XOR se používá na zkombinování proudu dat a pseudonáhodný proud bitů vytvořený z šifrovacího klíče, popř. algoritmu. Šifry jenž využívají XOR se nazývají *Vernamovy šifry*.

Jejich výhodou je snadná implementace. Jak hardwarově, tak softwarově je XOR jednoduchá operace. Při softwarové implementaci je však potřeba brát v úvahu, že posun jednotlivých bitu není velmi efektivní operace, když operace, které ji provádějí typicky pracují nad byty dat. Pokud se pracuje s binárními daty je potřeba data zorganizovat do skupin, které odpovídají velikosti slova daného procesoru[5, str. 88-89].

Nejznámějším příkladem proudové šifry je tzv. *One-time pad*. Jedná se o příklad bezpodmínečně bezpečného systému. Jedná se o takový systém, který útočník neprolomí ani s nekonečně velkým výpočetním systémem. One-time pad funguje na jednoduchém principu. Využívá bloky náhodných dat – *výplně* (angl. *Pad*). Výplň musí být minimálně stejně velká jako šifrovaná zpráva. Odesílatel zprávy použije pro každý bit zprávy a výplně funkci XOR, čímž vznikne zašifrovaný text. Příjemce použije stejnou výplň a funkci XOR k rozkódování zprávy.

Přestože je tento systém neprolomitelný, tak je z praktického hlediska naprosto nevyužitelný. První problém spočívá ve vygenerování opravdu náhodné výplně. Všechny dnes známe generátory jsou pouze pseudonáhodné, a tak existuje možnost, že by útočník mohl vygenerovat stejnou výplň. Druhý problém spočívá v synchronizaci výplně. Výplň musí být taky odeslána, ovšem jiným šifrováním. To už je ovšem z podstaty prolomitelné a nebezpečné a celý systém je tak silný, jak jeho nejslabší článek. Proto je smysluplnější využít právě tuto šifru[21, str. 197].

2.2.5 Blokové šifry

Jak již název napovídá, *blokové šifry* jsou skupinou šifrovacích algoritmů, které pracují s bloky fixní délky. Jestliže je zpráva větší než blok, tak se rozdělí na více bloků. Pokud je zpráva menší než jeden blok, tak je doplněna výplní. Dešifrování probíhá opět v blocích.

V praxi bylo prokázáno, že tento typ šifer není příliš bezpečný a je snadno prolomitelný. Slabinou je použití stejného klíče na všechny bloky. Proto bylo vytvořeno 5 základních *módů činnosti* blokových šifer, které se liší v použití základního šifrovacího bloku[8, str. 28].

ECB

ECB (*Electronic Cipher Book*) je základním módem pro všechny blokové šifry. Bloková šifra se chová jako překladový slovník. Pro každý blok otevřeného textu existuje právě jeden blok šifrovaného textu tak dlouho, dokud se nezmění klíč.

Útok na tento mód je velice jednoduchý. Útočník si je schopen vytvořit svůj vlastní slovník, a protože jsou bloky šifrovány na sobě nezávisle, tak může kdykoliv přenášený blok nahradit vlastním bez toho, aby to příjemce poznal[9, str. 9-16][10, str. 99-112][5, str. 79-86].

CBC

CBC (*Cipher Block Chaining*) mód řetězí všechny bloky za sebe, tím že kombinuje poslední zašifrovaný blok se současným nezašifrovaným blokem pomocí operace XOR. Teprve výsledek této operace se šifruje. Nevýhodou je stálé stejné šifrování prvního bloku. To se řeší tzv. inicializačním vektorem. Inicializační vektor bývá v praxi pseudonáhodné číslo, které se zkombinuje s prvním blokem pomocí XORu. Tento vektor se nemusí nijak tajit, ba dokonce se může připojit ke zprávě ve formě otevřeného textu, neboť tento vektor nijak nepomůže útočníkovi.

Dále ještě existuje nestandardizovaný a málo využívaný mód PCBC (Propagating Cipher Block Chaining), který vychází právě z CBC. Funguje stejně jako CBC, ale navíc je XORován ještě předchozí nešifrovaný text[9, str. 9-16][10, str. 99-112][5, str. 79-86].

CFB

CFB (*Cipher Feedback*) mód je rozdílný od CBC v tom, že samotná šifra je použita jako pseudonáhodný generátor místo toho, aby sloužila k šifrování a dešifrování. Šifrovaný text vzniká zkombinováním předchozího zašifrovaného bloku a současného bloku s otevřeným textem. Avšak oproti CBC, předchozí zašifrovaný blok prošel současným šifrováním, čímž bylo dosaženo onoho pseudonáhodného efektu. Pro první blok je opět použit inicializační vektor[9, str. 9-16][10, str. 99-112][5, str. 79-86].

OFB

OFB (*Output Feedback*) mód je velice podobný CFB módu, avšak s malinkou úpravou. Zatímco v CFB módu je vstupem pseudonáhodného generátoru výsledný zašifrovaný text pomocí XORu, tak v OFB je vstupem výsledek samotné blokové šifry. Je nutné podotknout, že OFB mód trpí stejnými základními nedostatky jako proudové šifry. Pro první blok je opět použit inicializační vektor[9, str. 9-16][10, str. 99-112][5, str. 79-86].

CTR

CTR (*Counter*) mód je opět velice podobný CFB módu tím, že využívá blokovou šifru jako pseudonáhodný generátor. Avšak na rozdíl od ostatních módu s pseudonáhodným generátorem nevytváří žádnou posloupnost. Díky tomu nepotřebuje znát žádný předchozí zašifrovaný blok a jeho implementace dosahuje větší rychlosti. Místo předchozího bloku je pro vstup generátoru použito počitadlo (angl. Counter) společně s inicializačním vektorem. Ačkoliv tento vektor není potřeba udržovat v tajnosti, jeho zašifrování dělá systém mnohem méně náchylnější k prolomení[9, str. 9-16][10, str. 99-112][5, str. 79-86].

2.3 Kryptoanalýza

Kryptoanalýza je věda zaobírající se luštěním a prolamováním šifer a hesel. Člověk jenž se takovou vědou zaobírá je nazýván *kryptoanalytik*. Výsledkem úspěšné kryptoanalýzy bývá samotný otevřený text, popř. klíč k jeho rozluštění. Nejlepší možnou variantou jakou lze v kryptoanalýze dosáhnout je stav, kdy kryptoanalytik je schopen vytvořit obecný algoritmus s jehož pomocí je schopný dešifrovat data bez znalosti šifrovacího klíče, popř. je schopen takový klíč vždy získat[25, str. 45].

Následující podkapitoly obsahují popis několika málo vybraných kryptoanalytických přístupů, jenž vedou k dešifrování zašifrovaného textu.

2.3.1 Útok hrubou silou

Útok hrubou silou (angl. Brute force attack) je takový útok, kdy se útočník snaží dešifrovat šifrovaný text zkoušením všech možných klíčů. Jedná se o nejprimitivnější druh útoku a k jeho provedení je potřebný nesmírně velký výpočetní výkon. Luštící počítač totiž prochází celý prostor všech možných klíčů[20, str. 7-9].

Součásti tohoto útoku je i problém zastavení. Zde je otázkou, jak má počítač poznat, že našel správnou kombinaci. Pokud je k otevřenému textu přiložen i kontrolní hash, je jednoduché takovou skutečnost ověřit. Problém ovšem nastává, pokud není. Zde je potřeba hledat známá slova, popř. jakákoliv smysluplná slova s porovnávaným slovníkem[22, str. 86-89].

2.3.2 Luštění se znalostí šifrovaného textu

Luštění se znalostí šifrovaného textu (angl. Ciphertext-only attack) je metoda, při níž máme k dispozici několik šifrovaných textů zašifrovaných pomocí stejného algoritmu a zároveň pomocí stejného klíče. Jedná se o jeden z nejčastějších druhů útoku, neboť většinou není problém získat velké množství šifrovaného textu. Analýzou zpráv a jejich společných znaků je pak odhalen klíč a díky tomu je umožněno čtení dalších zpráv[17, str. 72-88].

2.3.3 Luštění se znalostí otevřeného textu

Luštění se znalostí otevřeného textu (angl. Known-plaintext attack) je metoda velice podobná luštění se znalosti šifrovaného textu. Avšak kromě šifrovaných zpráv je známý i otevřený text. Cílem v takovém případě je zjistit klíč, aby bylo umožněno dešifrovaní dalších zpráv[15, str. 8-9].

2.3.4 Luštění se znalostí vybraných otevřených textů

Luštění se znalostí vybraných otevřených textů (angl. Chosen-plaintext attack) je metoda, kdy si kryptoanalytik sám může vybrat, které otevřené texty chce zašifrovat. Díky tomu má k dispozici otevřený text, tak i jeho šifrovanou podobu. S touto metodou může získat mnohem více informací o klíči než v předchozích případech[15, str. 8-9].

Existuje ještě varianta toho útoku známá jako Adaptivní metoda luštění se znalostí vybraných otevřených textů (angl. Adaptive-chosen-plaintext attack). V tomto případě si kryptoanalytik sám stanoví, které otevřené texty se mají zašifrovat na základě výsledku předchozího šifrovaní otevřeného textu. S touto metodou je možné analyzovat šifru mnohem rychleji[8, str. 31-33].

2.3.5 Luštění se znalostí vybraných šifrovaných textů

Luštění se znalostí vybraných šifrovaných textů (angl. Chosen-ciphertext attack) je velice netypický útok. Jedná se o případ, kdy si kryptoanalytik sám může zvolit, ke kterým šifrovaným textům dostane jejich otevřenou podobu. Tento přístup je například aplikovatelný pokud máme přístup k dešifrovacímu zařízení, avšak je potřeba převést otevřený text na šifrovaný[18, str. 117-121].

2.3.6 Útok postranním kanálem

Útok postranním kanálem (angl. Side-channel attack) je souhrnné označení pro jakýkoliv útok, jenž se nesnaží najít jakoukoliv matematickou slabinu v algoritmu, ale využívá nedostatků ve fyzické implementaci počítačového systému. Útoky postranními kanály jsou nejčastěji založené na časových analýzách, odběrových analýzách a elekromagnetických analýzách[12, str. 402-410].

Kapitola 3

Aktuální standardy v kryptografii

V této kapitole se budu zabývat aktuálně používanými algoritmy v oblasti kryptografie. Vždy je zde zastoupen jeden algoritmus pro jedno šifrovací odvětví. Cílem je seznámit čtenáře s hlubším pochopením avizovaných algoritmů, jejich implementace a mírou bezpečnosti při jejich použití.

3.1 AES

AES (Advanced Encryption Standard) je blokový šifrovací algoritmus využívající symetrický klíč. Jedná se o první šifrovací algoritmus, jenž je dostupný pro širokou veřejnost a zároveň byl uznán NSA (National Security Agency) za vhodný k šifrování jejich nejtajnějších dokumentů.

S končící platností šifrovacího standardu DES (Data Encryption Standard) roku 1998 bylo nutné vytvořit nový bezpečnostní standard. Proto v NIST (National Institute of Standards and Technology) vyhlásil veřejnou soutěž do níž se zapojilo 15 oficiálních kandidátů, z nichž 5 se probojovalo do finále. Na rozdíl od algoritmu DES, který byl speciálně navržen pouze pro hardwarovou implementaci, musel algoritmus AES být efektivně implementovatelný jak hardwarově, tak softwarově. Dále bylo vyhlášeno, že AES musí být bloková šifra, která využívá bloky o délce 128 bitů a podporuje klíče o délce 128, 192 a 256 bitů. Jako nový standard byla vybrána šifra Rijndael. Její název vznikl přesmyčkou jmen jejích autorů Joana Daemena a Vincenta Rijmena, jenž pocházeli z Belgie[10, str. 139-140].

Popis algoritmů

Algoritmus AES mapuje otevřený text na matici o velikosti 4x4. Matice je sloupcově dominantní, což znamená, že nejvýznamnější byte je namapován na pozici a_{00} a druhý nejvýznamnější byte na pozici a_{10}. Obdobně je namapován šifrovací klíč na matici o velikosti 4x*délka klíče dělena 32*, což znamená velikost 4, 6 a 8 pro klíče o velikosti 128, 192 a 256 bitů. Počet iterací potřebných k zašifrování a odšifrování textu je založen na velikosti klíče. Pro 128 bitový klíč se jedná o 10 iterací, pro 192 bitový klíč se použije 12 iterací a 256-bitový využívá 14 iterací.

Algoritmus funguje následujícím způsobem:

1. Expanze klíče - podklíče jsou odvozeny z klíče šifry užitím Rijndael programu

2. Inicializační část

(a) Přidání podklíče - každý byte stavu je zkombinován s podklíčem za pomoci operace xor nad všemi bity

3. Iterace

   (a) Záměna bytů - nelineární nahrazovací krok, kde je každý byte nahrazen jiným podle vyhledávací tabulky
   (b) Prohození řádků - provedení kroku, ve kterém je každý řádek stavu postupně posunut o určitý počet kroků
   (c) Kombinování sloupců - zkombinuje čtyři byty v každém sloupci
   (d) Přidání podklíče

4. Závěrečná část

   (a) Záměna bytů - nelineární nahrazovací krok, kde je každý byte nahrazen jiným podle vyhledávací tabulky
   (b) Prohození řádků - provedení kroku, ve kterém je každý řádek stavu postupně posunut o určitý počet kroků
   (c) Přidání podklíče

Záměna bytů je provedena pro každý z 16 prvků matice. Byty jsou nahrazovány pomocí 8-bitového pozměňovacího boxu, tzv. Rijndael S-box, který slouží jako vyhledávací tabulka. Díky tomuto kroku je v algoritmu zajištěna nelinearita.

Prohození řádků provádí bytovou rotaci vlevo na řádcích pole, jenž je výsledkem záměny bytů. První nultý řádek není prohozen vůbec. Řádek jedna je rotován o jeden byte, řádek dva o dva byty a poslední řádek je rotován o tři byty. Tato operace má za následek, že hodnoty které byly předtím zarovnány ve sloupcích jsou nyní zarovnány pomocí levé diagonály. To je stejný výsledek, jako kdyby data byla zaměněna ve Feistelově síti. A i přesto, že AES Feistelovou síť nevyužívá, je schopen zašifrovat celý blok v jednom kroku, což v konečném důsledku snižuje celkový počet kroků v provedení celého algoritmu[7, str. 8-16] .

Při operaci kombinování sloupců je každý sloupec pole maticově vynásoben s výslednou maticí z operace prohození řádků. Díky tomu všechny vstupní byty ovlivňují všechny výstupní byty a je tím zajištěna další nelineární ochrana.

Poslední operací je přidání pod-klíče. V ní je zkombinován výstup operace kombinování sloupců a příslušného pod-klíče pomocí bitové operace XOR. Každý pod-klíč musí mít stejnou velikost jako samotná matice. Výstup poslední operace přidání pod-klíče je i zároveň výstupem finálním[20, str. 87-112].

Známe útoky

V současnosti neexistuje žádný známý útok, který by prolomil plně funkční, správně implementovaný algoritmus AES. Existovalo mnoho pokusů, avšak žádný úspěšný.

V roce 2011 byl publikován doposud nejlepší útok na plně funkční AES za účelem obnovení celého klíče. Za tímto útokem stojí Andrey Bogdanov, Dmitry Khovratovich a Christian Rechberger. Jedná se o tzv. bicyklický útok. Pro plné obnovení klíče o délce 128-bitů je potřeba $2^{126.2}$ operací. Pro AES-192 se jedná o $2^{190.2}$ potřebných operací a pro AES-256 se jedná o $2^{254.6}$ operací. Tento výsledek byl později ještě o něco málo vylepšen[4, str. 1-3] .

3.2 RSA

RSA je jedna z prvních asynchronních šifer s veřejným klíčem. Ačkoliv jeho patent vypršel 21. září 2000, tak se stále jedná o jeden z nejpopulárnějších asynchronních šifrovacích algoritmů vůbec a při dostatečné velikosti klíče je stále považován za bezpečný.

V roce 1977 byl systém RSA navržen Ronem Rivestem, Adi Shamirem a Leonardem Adlemanem z MIT (Massachusetts Institute of Technology). Z jejich jmen je také odvozen název algoritmů. Vývoj byl mnohonásobně složitější oproti symetrickému šifrování, protože se jedná o matematicky mnohem náročnější problém. Systém si nechalo patentovat MIT v roce 1983 ve Spojených Státech Amerických. Platnost patentu vypršela v roce 2000[6, str. 1].

Popis algortimu

Algoritmus je založen na obtížnosti faktorizace velkých čísel a skládá se ze dvou částí:

1. Generování klíčů
2. Šifrování/Dešifrování

Pro vygenerováni klíče je nejdříve potřeba zvolit dvě náhodná čísla p a q, která jsou zároveň i prvočísla. Jejich vynásobením získáme

$$n = p * q$$

Následně je potřeba zjistit počet čísel mezi 1 a $n - 1$, která jsou relativními prvočísly čísla n. Tato funkce je známá jako Eulerova funkce a značí se ϕ. Počet relativních prvočísel čísla n se tedy spočítá jako

$$\phi(n) = (p - 1) * (q - 1)$$

Následně se náhodně vybere takové celé číslo, které splňuje podmínku $0 < b < \phi(n)$ a zároveň pro které platí, že nejmenší společný dělitel čísla ϕ a čísla b je roven 1. Nakonec se spočítá číslo a jako

$$a = b^{-1} \mod \phi(n)$$

Tímto je generování klíčů dokončeno a veřejný klíč získáme jako

$$K_{PUB} = (n, b)$$

a privátní klíč jako

$$K_{PR} = (p, q, a)$$

Následně je potřeba se důkladně zbavit čísel p a q, neboť jejich získáním lze jednoduše oba klíče vygenerovat.

Aby jsme otevřený text x zašifrovali na šifrovaný text y použijeme následující funkci:

$$y = x^b \mod n$$

Dešifrování zašifrovaného textu y na otevřený text x provedeme následující funkcí:

$$x = y^a \mod n$$

[10, str. 223-224][21, str. 203-205].

Známé útoky

Aby mohl být proveden útok za účelem úplného nalezení klíče, je potřeba aby byl otevřený text x zašifrován veřejným klíčem tak, že $y = x^b \mod n$ a byly vyzkoušeny všechny privátní klíče v rozsahu $0 \leq a < \phi(n)$ a nalezen ten, jenž splňuje, že $x = y^a \mod n$. Aby byla zajištěna bezpečnost proti takovému útoku musí být modulo n zvoleno tak, že $n > 2^{500}$. Takto dostatečně velké modulo zajišťuje, že útok je v praxi neproveditelný.

Další existující útok na algoritmus RSA je založen na faktorizaci čísel. Nejprve je nalezeno $\phi(n)$ vzhledem k $y = x^b \mod n$ a veřejnému klíči. Následně je spočítáno a jako $a = b^{-1} \mod \phi(n)$. Když je a známo je otevřený text x rozšifrován jako $x = y^a \mod n$. Avšak spočítání $\phi(n)$ není triviální záležitostí a ve své složitosti je stejné jako integrace modula n.

Útoky založené na principu tohoto algoritmu ovšem dosahují tak velkých výpočetních časů, že nejsou reálně použitelné[24, str. 665-671] .

3.3 SHA

SHA je skupina hashovacích algoritmů, jenž jsou vydávány pod záštitou NIST (National Institute of Standards and Technology). Do této skupiny se aktuálně řadí 4 algoritmy. Jedná se o SHA-0, SHA-1, SHA-2 a SHA-3. Tato část se bude nejvíce zaobírat algoritmem SHA-3, který je nejvíce rozdílný od zbytku skupiny.

SHA-3 je na rozdíl od zbytku algoritmu skupiny SHA úplně jiný. Tyto zbylé algoritmy založeny na Merkle-Damgardově principu a vycházejí z hashovací funkce MD4 a MD5. SHA-3 naproti tomu pochází ze skupiny Keccakových algoritmů. Za jejim vznikem stojí skupina kryptografů v seskupení: Guido Bertoni, Joan Daemen, Michaël Peeters a Gilles Van Assche. Joan Daemen je mimo jiné taky spoluautorem šifry Rijndael, která slouží jako základ pro AES.

V roce 2006 uspořádala NIST soutěž ve vytvoření nové hashovací funkce, která by se stala standardem známým jako SHA-3. Soutěž byla veřejná, neboť v NIST vznikaly obavy o bezpečnost SHA-2, protože existovaly úspěšné útoky na SHA-0 a SHA-1. Z tohoto důvodu chtěli další alternativu v oblasti hashovacích funkcí, neboť SHA-2 je založena na stejném principu jako SHA-1. 2. října 2012 byl Keccak vyhlášen jako vítěz soutěže. 5. srpna 2015 NIST prohlásilo SHA-3 za nový hashovací standard.

Popis algoritmu

Hashovací funkce SHA-3 je založena na principu Sponge funkce[3] (dosl. Houba). Velikost vnitřního stavu je určena pomocí šířky permutace Keccak-f, která má 1600 bitů. Dále je potřeba mít parametry bitového toku r a kapacity c, jejichž součet je roven velikosti vnitřního stavu. Velikost vnitřního bloku r má velikost $1600 - c$. c je definováno jako dvojnásobek výsledné délky hashe. Samotný algoritmus má 2 fáze:

1. Absorbovací – Na začátku je vstupní blok XORován s prvními r bity vnitřního stavu. Na výsledek této operace se použije permutace Keccak-f. Celá fáze se opakuje, dokud nejsou zpracovány všechny bloky.

2. Vymačkávací – Na výstup je použito prvních r bitů vnitřního stavu. Na ty je použita permutace Keccak-f. Celá fáze se opakuje, dokud není vytvořen požadovaný počet bloků.

Známe útoky

U hashovacích funkcí neřešíme zabezpečení proti útoku, jako zabezpečení proti kolizi. Kolize je speciální případ, kdy dva vstupy hashovací funkce vyprodukují totožný výstup. Tento stav je umožněn díky tomu, že výstup hashovací funkce má neměnnou délku, zatím co její vstup je libovolný. Mezi nejznámější útoky snažící se nalézt kolizi hashovací funkce patří tzv. *narozeninový útok* (angl. *Birthday attack*) a *Preimage attack*.

Bezpečnost algoritmu SHA-3 proti kolizi je definována jako $min(c/2, n/2)$ a bezpečnost proti Preimage útoku je definována jako $min(c/2, n)$, kde n je velikost výstupu[28, str. 468-471].

Kapitola 4

Existující řešení

V této kapitole se zaměřím na již existující řešení, která se zaměřují na ochranu dat obdobným způsobem jako aplikace vytvořená v rámci této práce. Tedy aplikace nebo knihovny, jenž ukládají citlivé identifikační údaje jako hesla, certifikáty, apod. a dále jsou nezávisle na platformě.

4.1 Keychain

Keychain (česky *klíčenka*) je software od společnosti *Apple*, který slouží jako zabezpečené uložiště hesel, certifikátů, privátních klíčů a poznámek. Tento software je dostupný pouze na platformě Mac OS, a to od verze Max OS X a všech následujících.

Keychain funguje na principu jednoho hlavního hesla. Při každém pokusu o uložení nových dat nebo jejich získání je uživatel vyzván, aby toto heslo zadal. Heslo je primárně shodné s heslem uživatelského účtu, avšak lze ho změnit na jiné pro ještě větší bezpečnost.

Pro tento software existuje taky uživatelské rozhraní nazvané *Keychain Acces*. Toto rozhraní umožňuje zpravovat všechna uložená data. Je možné zde mazat a vytvářet nové hesla a poznámky, což může být jediný způsob, jak něco opravit v případě problémů.

Kromě Keychain existuje ještě *iCloud Keychain*, který má úplně stejný úkol, ale slouží pro operační systémy iOS. Byl představen ve verzi iOS 7.0.3. a od té doby je jeho běžnou součástí. Tento systém také umožňuje navrhování nových bezpečných hesel, avšak pouze pro webový prohlížeč *Safari* od společnosti Apple. Systém taky zálohuje hesla mezi jednotlivými zařízeními uživatele pomocí služby iCloud.

Podle oficiální dokumentace[2] využívá systém zabezpečení AES se 256-bitovým klíčem v CTR módu.

4.2 Crypto++

Crypto++ je knihovna pro C++ implementující širokou škálu různých šifrovacích algoritmů, hashovacích funkcí a dokonce některé zastaralé způsoby šifrovaní pro zpětnou kompatibilitu. Knihovna je dostupná pro platformy iOS, Windows a většinu Unixových distribucí a běží pod distribucí a licencí Boost Software License 1.0, avšak jednotlivé soubory jsou bez licence jako volné dílo.

Crypto++ je aktivně používanou součástí C++. Důkazem je poslední aktualizace z 22. ledna 2018 na verzi 6.0.

Algoritmus	MiB/Second	Cycles Per Byte	Cycles to Setup Key and IV
AES/CTR (128-bit key)	4525	0.57	598
AES/CTR (192-bit key)	3845	0.67	567
AES/CTR (256-bit key)	3340	0.77	630
AES/CBC (128-bit key)	1073	2.40	459
AES/CBC (192-bit key)	920	2.80	431
AES/CBC (256-bit key)	805	3.20	483
AES/OFB (128-bit key)	994	2.59	596
AES/CFB (128-bit key)	1048	2.46	644
AES/ECB (128-bit key)	5006	0.51	186
AES/GCM	2789	0.92	1008
AES/CCM (128-bit key)	864	2.98	709
AES/EAX (128-bit key)	864	2.98	922

Tabulka 4.1: Výkonnostní test šifry AES v implementaci Crypto++ 6.0.

Algoritmus	Milliseconds/Operation	Megacycles/Operation
RSA 1024 Encryption	0.01	0.03
RSA 1024 Decryption	0.23	0.63
RSA 2048 Encryption	0.03	0.07
RSA 2048 Decryption	1.03	2.78

Tabulka 4.2: Výkonnostní test šifry RSA v implementaci Crypto++ 6.0.

Implementace AES v Crypto++

V tabulce 4.1 lze nalézt výsledky z výkonnostního testu implementace šifry AES v knihovně Crypto++ ve verzi 6.0. Test byl prováděn na procesoru Skylake Core-i5 s obnovovací frekvencí 2.7 GHz[1].

Implementace RSA v Crypto++

Pro urychlení mnoha implementací systému RSA je často používána čínská věta o zbytcích. Dále jsou využívána další jak hardwarová, tak softwarová vylepšení. Avšak i přes jejich použití, dosahuje asymetrické šifrování pomocí RSA o jeden až tři řady pomalejší rychlosti než symetrické šifrování jako třeba AES.

Protože je RSA deterministický šifrovací systém, je možné na něj zkoušet útok pomocí vybraných otevřených textů, které útočník zašifruje pomocí veřejného klíče, a pak je mezi sebou navzájem porovnává. Aby se zabránilo tomuto druhu útoku, je často implementováno tzv. schéma doplnění. To zajistí, že zpráva bude posunuta o náhodné číslo a její šifrování vytvoří velké množství možností zašifrovaného textu.

V tabulce 4.2 lze nalézt výsledky z výkonnostního testu implementace šifry RSA v knihovně Crypto++ ve verzi 6.0. Test byl prováděn na procesoru Skylake Core-i5 s obnovovací frekvencí 2.7 GHz[1].

Algoritmus	MiB/Second	Cycles Per Byte
SHA-1	540	4.76
SHA-256	275	9.38
SHA-512	377	6.83
SHA3-224	280	9.21
SHA3-256	264	9.74
SHA3-384	203	12.69
SHA3-512	141	18.27

Tabulka 4.3: Výkonnostní test šifry RSA v implementaci Crypto++ 6.0.

Implementace SHA v Crypto++

Vzhledem k faktu, že algoritmus využívá pouze bitové operace XOR, AND, NOT a rotace, tak je jeho hardwarová implementace velice jednoduchá a algoritmus dosahuje vysoké propustnosti.

Softwarová implementace oproti tomu dovoluje využít jen dvoucestný paralelismus a algoritmus díky tomu nedosahuje stejně dobrých výsledků jako u implementace hardwarové. Avšak algoritmus je optimalizován pro 64bitové procesory.

V tabulce 4.3 lze nalézt výsledky z výkonnostního testu implementace hashovacích funkcí SHA v knihovně Crypto++ ve verzi 6.0. Test byl prováděn na procesoru Skylake Core-i5 s obnovovací frekvencí 2.7 GHz[1].

4.3 CryptoAPI

Microsoft Cryptographic API je aplikační rozhraní pro operační systémy Microsoft Windows poskytující vývojářům možnosti, jak zabezpečit své aplikace pro platformu Windows. Jedná se o kolekci dynamických knihoven představených ve verzi Windows NT 4.0 a v každé následující verzi vylepšených.

CryptoAPI obsahuje jak symetrické šifrování, tak asymetrické. Dále nabízí šifrování a dešifrování dat za pomoci certifikátů a pseudonáhodný generátor.

Kromě CryptoAPI existuje i jeho vylepšená verze známá jako Cryptography API: Next Generation (CNG) představená ve verzi Windows Vista. Nabízí širší škálu algoritmů a celou řadu novějších algoritmů, jenž jsou součástí NSA (National Security Agency). CNG je zpětně kompatibilní s CryptoAPI. Mimo jiné, CNG funguje i v Kernel módu a nabízí možnost kryptografie nad eliptickými křivkami. Dalším významným posunem je použití AES jako blokové šifry namísto DES, jenž byla dávno prolomena.

Kapitola 5

Návrh knihovny pro ukládání citlivých dat

V předchozí kapitole jsme porovnali některé možné přístupy a zhodnotili dosavadní existující řešení našeho problému. V následující kapitole bude popsán návrh funkčnosti knihovny a jejího rozhraní.

5.1 Cílová skupina

Knihovna pro tvorbu bezpečného uložiště by měla sloužit všem vývojářům pracujícími s jakýmikoliv daty, které by se neměly dostat do rukou jakékoliv cizí osobě – ať se jedná o potencionálního útočníka nebo samotného uživatele.

Využití najde u tvorby menších projektů, u kterých je potřeba skrýt soubory s nastavením, uložená data, popř. jakákoliv jiná data, ke kterým by se uživatel aplikace neměl dostat. Avšak uplatnění lze najít i u větších projektů, kde může jednoduše chránit uložená hesla, certifikáty, autorizační klíče a podobné jiné zabezpečovací mechanismy.

5.2 Požadavky na knihovnu

Většina požadavků vyplývá již ze základních principů přístupnosti a uživatelské přívětivosti. Knihovna by měla ale především splňovat všechny tyto požadavky:

- Vysoká bezpečnost bez znalosti zdrojového kódu
- Jednoduché aplikační rozhraní
- Dostupnost na Windows a iOS
- Data dostupná pouze na jednom zařízení jednomu uživateli
- Ochrana dat před náhodným smazáním
- Automaticky generované heslo o které se nemusí programátor starat
- Ukládání dat v mnoha typech
- Možnost kategorizace dat do modulů

- Tvorba, upravování a mazání modulů
- Bezpečná práce s více vlákny
- Rychlá implementace
- Sada unit testů jako součást řešení

Aplikační rozhraní knihovny by mělo obsahovat tyto funkce:

- uložitData(string klíč, data)
- získatData(string klíč, data)
- smazatData(string klíč)
- vytvořitModul(název)
- uložitDataDoModulu(string názevModulu, string klíč, data)
- získatDataZModulu(string názevModulu, string klíč, data)
- smazatModul(string název)

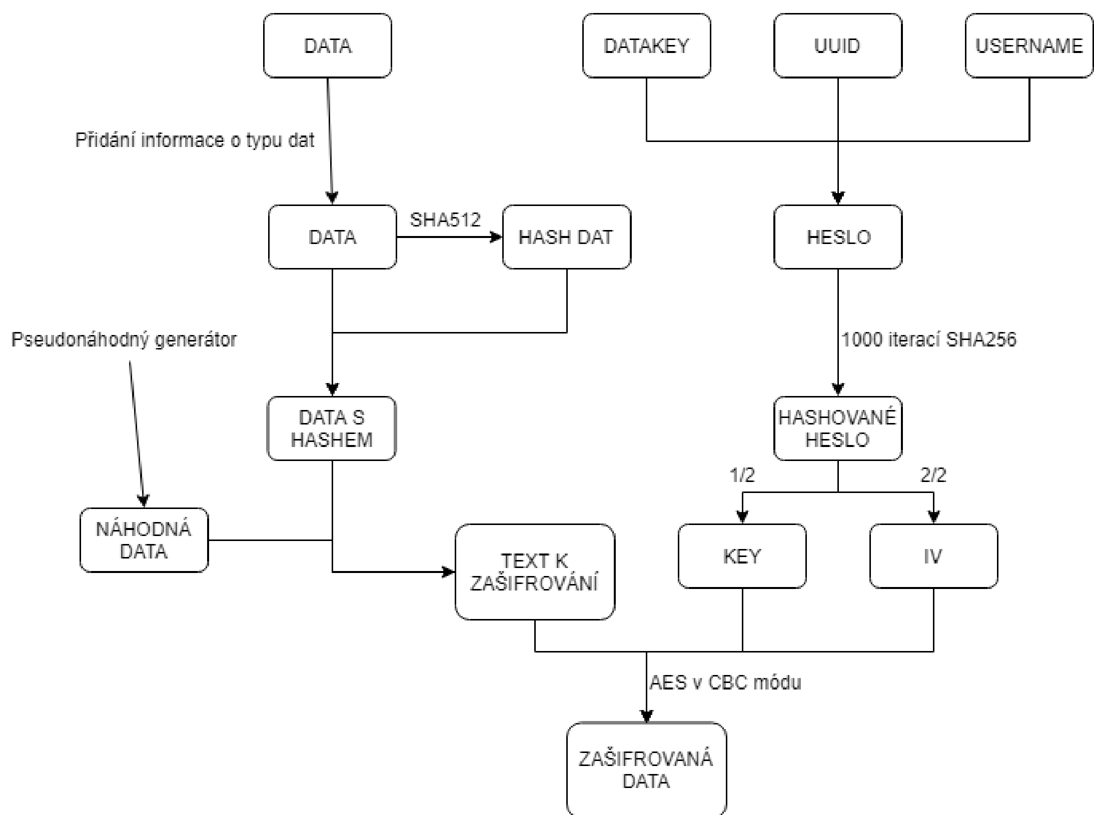
5.3 Šifrování dat

Pro opravdu bezpečné šifrování dat bylo využito několik známých principů. Veškerá data jsou nejprve převedena na typ String a je k nim přidána informace o jejich původním typu. Nad tímto řetězcem je následně zavolána hashovací funkce SHA512, jejiž výsledek je poté připojen k datům. Tento hash bude později sloužit k ověřovacím účelům. Obdobná metoda je využívána u elektronických podpisů. Dále je vygenerován blok náhodných znaků, který je připojen k datům, již obsahujících informaci o typu dat a hash. Náhodná data slouží jako obrana proti slovníkovým útokům. Tento celek je následně zašifrován pomocí šifry AES v CBC módu. Šifra a její mód byly vybrány z důvodu vysoké spolehlivosti a obstojné efektivitě.

Šifra AES v CBC módu vyžaduje počáteční klíč a inicializační vektor. Protože požadavky na modul vyžadují, aby programátor nebo uživatel nebyli nuceni heslo zadávat, je nutné tyto dvě entity vygenerovat. Specifikace také udává, že uživatel by měl být schopen přistoupit k datům pouze na stejném zařízení pod stejným uživatelským účtem. Jako identifikace počítače bylo vybráno jeho UUID, jenž by mělo byt specifické pro všechny zařízení, zároveň by ho však každé zařízení mělo obsahovat. Pro identifikaci uživatele bylo zvoleno jeho uživatelské jméno. Jako třetí část klíče je identifikační jméno dat, které musí programátor zadat při jejich ukládání. Kombinace těchto tří prvků slouží jako unikátní heslo pro každá uložená data. Heslo je ještě následně pomoci Key Derivation Function zahashováno tisícem iterací. První půlka hesla slouží jako klíč a druhá půlka slouží jako inicializační vektor. Celý postup je graficky znázorněn na obrázku 5.1.

5.4 Dešifrování dat

Proces dešifrování je kompletně opačný k procesu šifrování. Uložená zašifrovaná data jsou nejprve rozšifrována pomocí šifry AES v CBC módu. Klíč a inicializační vektor jsou vytvořeny naprosto stejně jako v případě šifrování.



Obrázek 5.1: Návrh šifrování dat v knihovně PISSD.

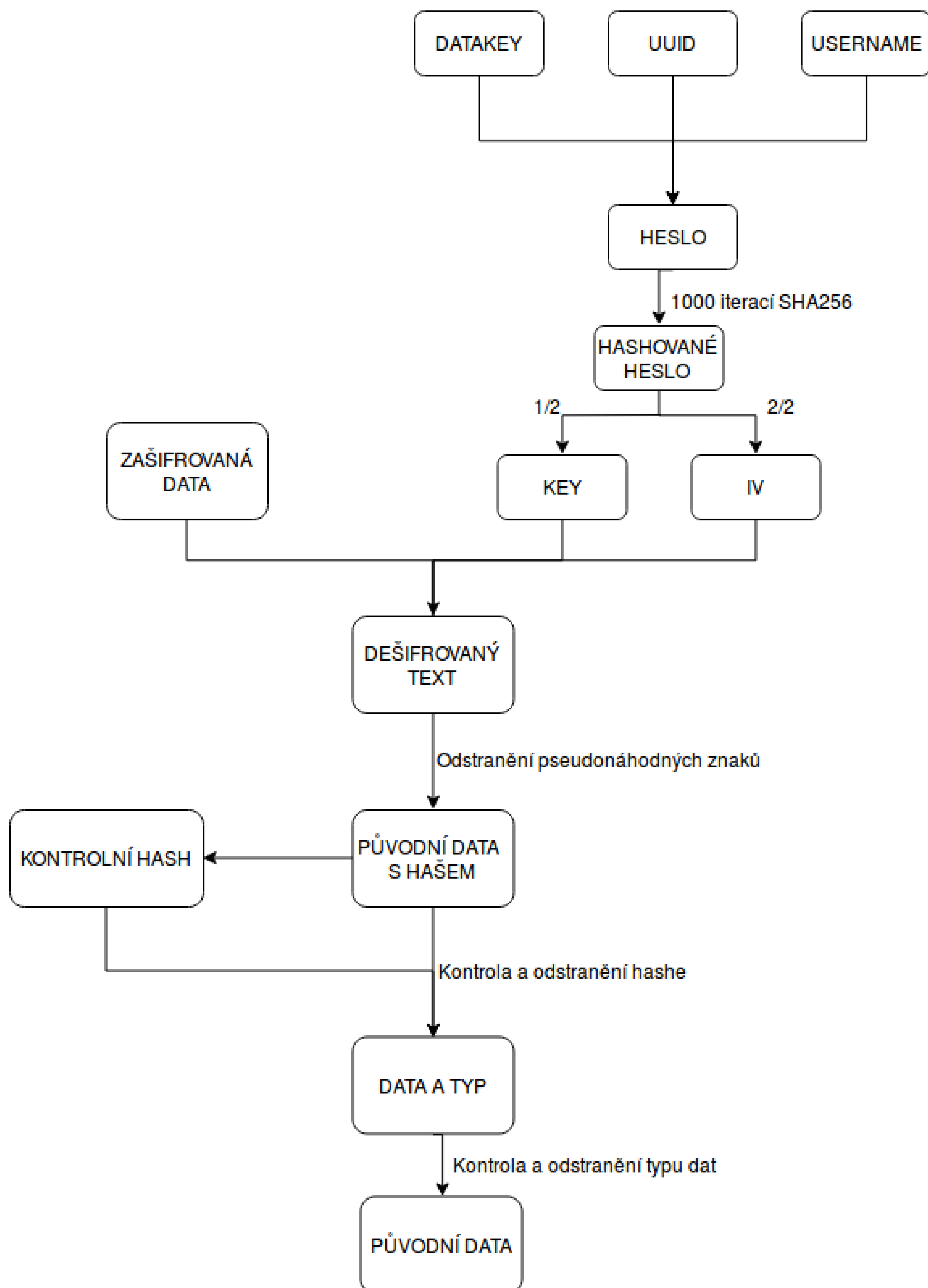
Od získaného řetězce je nejprve odebrána část pseudonáhodných znaků. Dále je vyjmut hash a uložen bokem. Nad zbylými daty je provedena opět hashovací funkce a její výsledná hodnota je porovnána s původním hashem. Nakonec je provedena konverze na původní typ dat. Celý postup je graficky znázorněn na obrázku 5.2.

5.5 Ukládání dat s redundancí

Pro potřeby knihovny PISSD byl navržen formát jkl. Jedná se o prostý znakový formát, sloužící výhradně pro odlišení souborů knihovny PISSD od zbytku souborového systému. Každé volání knihovny za účelem uložení dat, vytvoří nový soubor. Tento přístup slouží k zpomalení útočníka, který musí rozšifrovávat každý soubor zvlášť, a nestačí mu rozluštit pouze jeden soubor obsahující všechna data.

Aby bylo redukováno riziko ztráty dat, např. neopatrným zacházením ze strany uživatele nebo výpadku ze strany pevného disku, byl použit systém trojitého ukládání dat. Všechna uložená data jsou uložená na disk do třech různých míst. To ve výsledku znamená, že v systému existují tři identické kopie zašifrovaných dat. Tyto kopie nejenže slouží jako zálohy, ale také ochrana proti změně. Pokud jedna položka nebude souhlasit s ostatními, programátor se to dozví a bude záležet na něm, jak s touto informací naloží.

Všechny soubory a vzniklé složky jsou na disk ukládány jako skryté. Tento krok nemá žádnou bezpečnostní funkci, ale slouží čistě k tomu, aby se běžný uživatel vůbec nesetkal s touto strukturou, nebo dokonce ji sám neodstranil.



Obrázek 5.2: Návrh dešifrování dat v knihovně PISSD.

Kapitola 6

Implementace knihovny PISSD

V této kapitole budou popsány externí knihovny použité při implementaci a některá úskalí a specifika, která se vyskytla v průběhu tvorby této práce. Dále take bude popsána sada unit testů, jenž vznikla současně s knihovnou PISSD. Z externích knihoven se jedná o knihovnu boost, především pak její část filesystem, jenž posloužila pro práci se soubory a složkami, a knihovnu Crypto++, která slouží pro práci s šifrovacími algoritmy. Knihovna byla vyvinuta v jazyce C++ souběžně na operačním systému Windows 10 a macOS High Sierra. Primární použití knihovny PISSD je zaměřeno především na tyto dva systémy.

6.1 Boost

Boost je sada open source knihoven pro programovací jazyk C++. Knihovna je distribuována pod licenci Boost Software License. Za autory knihovny jsou považování Beman Dawes a David Abrahams.

Pro práci se soubory a celým souborovým systémem byla využita část knihovny Boost nazývánà Filesystem. Konkrétně byly využity pro vyhledávání klíčů v datech a odstraňování již nepotřebných částí. Podstatnou částí byla funkce BOOST_FOREACH, jenž sehrála velkou roli v procházení souborové struktury a nacházení hledaných položek. Kompletní výčet použitých funkcí a jejich stručných popisů lze nalézt v tabulce 6.1.

6.2 Crypto++

Crypto++ je volně dostupná knihovna pro programovací jazyk C++ obsahující implementaci velkého množství kryptografických algoritmů. Knihovna je distribuovaná pod licenci Boost Software License. Autorem knihovny je Wei Dai.

Knihovna Crypto++ byla využita v knihovně PISSD pro všechny úkony související s šifrováním, dešifrováním, hashováním a tvorbu pseudonáhodných znaků. Hlavní využití

Funkce	Popis
filesystem::remove	Metoda odstraní vybraný soubor nebo složku
filesystem::remove_all	Metoda odstraní všechny soubory a složky na zadané cestě
BOOST_FOREACH	Upravený for cyklus využitý pro iteraci po všech souborech

Tabulka 6.1: Použité funkce z knihovny Boost.

Třída	Popis
AES::Encryption	Instance blokové šifry AES k šifrování
CBC_Mode_ExternalCipher::Encryption	Objekt módu používající externí šifru k šifrování
AES::Decryption	Instance blokové šifry AES k dešifrování
CBC_Mode_ExternalCipher::Decryption	Objekt módu používající externí šifru k dešifrování
StreamTransformationFilter	Objekt umožňující symetrické šifře účastnit se proudového zpracování
StringSink	Objekt sloužící jako vyústění proudového zpracování šifry

Tabulka 6.2: Použité funkce z knihovny Crypo++ k šifrování a dešifrování.

Třída	Popis
SHA512	Třída sloužící k hashování pomocí algoritmu SHA512
StringSource	Třída sloužící jako zdroj pro pole znaků a stringů
HashFilter	Třída počítající hash na základě požadovaného algoritmu
Base64Encoder	Třída sloužící jako kodér převádějící binární data do posloupnosti tisknutelných znaků

Tabulka 6.3: Použité funkce z knihovny Crypo++ k hashování.

knihovny Crypto++ se nachází v implementaci kryptografického algoritmu AES. Použité třídy z této knihovny lze nalézt v tabulce 6.2.

Další významnou částí využití knihovny Crypto++ je hashovací funkce. Použité metody týkající se hashovací funkce lze nalézt v tabulce 6.3.

Knihovna Crypto++ byla mimo jiné také využita pro generování pseudonáhodných znaků a šifrování hesel pomocí KDF (Key Derivation Function). Použité objekty a metody lze nalézt v přiložené tabulce 6.4.

6.3 Knihovna PISSD

Knihovna PISSD byla vytvořena jako zastřešení veškeré funkcionality celého modulu. Aby práce s modulem byla jednodušší, obsahuje knihovna pouze jedinou třídu se kterou musí programátor přijít do styku a to SecureDataStorage. Instance této třídy je pak jediným

Třída	Popis
SHA512	Třída sloužící k hashování pomocí algoritmu SHA512
StringSource	Třída sloužící jako zdroj pro pole znaků a stringů
HashFilter	Třída počítající hash na základě požadovaného algoritmu
Base64Encoder	Třída sloužící jako kodér převádějící binární data do posloupnosti tisknutelných znaků

Tabulka 6.4: Použité funkce z knihovny Crypo++ k hashování.

možným způsobem, jak s modulem pracovat. Třída se stará o veškerou práci s šifrováním a dešifrováním dat a jejich ukládáním na disk. Třída SecureDataStorage nevyužívá žádné jiné zanořené podtřídy. Jejím úkolem je rovněž se starat o vytváření a mazání struktur do níchž můžou být data uložena.

Z důvodu přenositelnosti a kompilace knihovny na různých operačních systémech byl pro kompilaci knihovny zvolen program CMake, který byl pro tyto účely navržen a je široce využíván. V knihovně PISSD je vyžadována jeho minimální verze 3.9.

Pro správu verzí knihovny byl použit systém GitHub. Zdrojové soubory jsou dostupné na adrese: https://github.com/Tasarak/libPISSD.

6.4 Uložení dat

Při implementaci bylo potřeba vyřešit problém, kam se budou soubory ukládat z důvodu různých operačních systémů a jejich souborových systémů. Bylo nutné vybrat tři různá umístění, která běžně uživatel nepoužívá, avšak jsou dostupná bez administrátorských práv a běžně se na systému vyskytují. Pro operační systém Windows byly zvoleny tyto cesty:

- C:/User/AppData/Roaming/PISSD
- C:/User/AppData/Local/PISSD
- C:/User/Documents/PISSD

Pro operační systémy pod distribucí Apple jsou zvolené cesty následující:

- /Users/username/.config/.PISSD
- /Users/username/Documents/.PISSD
- /Users/username/Library/.PISSD

Všechny zvolené cesty by měly vyhovovat výše uvedeným požadavkům. Pro operační systémy Windows je navíc potřeba nastavit vytvořené složce skrytý atribut, aby byla v systému zobrazená jako skrytá.

6.5 Dešifrování dat

Reálná implementace dešifrování uložených dat v sobě skrývá několik úskalí navíc oproti předloženému návrhu.

Nejprve je potřeba načíst všechny uložené soubory. Pokud nejsou dostupné všechny tři, je třeba k nim přistupovat jinak než v případě, že jsou nalezeny všechny. Jestli jsou nalezeny všechny tři, je porovnán jejich zašifrovaný obsah. Pokud minimálně dva jsou stejné, je práce s nimi považována za bezpečnou. Následně jsou data rozšifrována, porovnány hashe a zjištěno, jestli data jsou stejného typu, jak je požadováno. Pokud data vyhovují všem výše uvedeným specifikům, tak se dočasně uloží jako možná data. Následně se zjišťuje, jestli alespoň dva dešifrované texty jsou stejné. Pokud ano, tak se navrátí jako výsledek.

V případě, že nejsou dostupné všechny tři požadované soubory, odpadne kontrola na stejný zašifrovaný text, avšak uživatel je upozorněn na skutečnost, že něco není v pořádku návratovým kódem funkce.

Scénář	Popis
Store and Retrieve String	Test vytvoří instaci knihovny, pokusí se uložit krátký text ve formátu string a následně se jej pokusí opět získat
Store and Retrieve Double	Test vytvoří instaci knihovny, pokusí se uložit desetinné číslo ve formátu double a následně se jej pokusí opět získat
Store and Retrieve Float	Test vytvoří instaci knihovny, pokusí se uložit desetinné číslo ve formátu float a následně se jej pokusí opět získat
Store and Retrieve Int64	Test vytvoří instaci knihovny, pokusí se uložit celé číslo ve formátu int64 a následně se jej pokusí opět získat
Store and Retrieve Bool	Test vytvoří instaci knihovny, pokusí se uložit proměnné typu bool nabývající obou možných hodnot a následně se je pokusí opět získat
Delete Stored Data	Test vytvoří instanci knihovny a vymaže soubory vzniklé v předchozích testech
Create and Remove Module	Test vytvoří instaci knihovny, pokusí se vytvořit modul a následně se jej pokusí odstranit
Store/Retrieve Data to/from Module	Test se pokusí do vytvořeného modulu uložit data všech typů (string, double, float, int64, bool) a následně se jej pokusí získat zpět
Get Keys	Test se nejprve pokusí navrátit všechny moduly, následně všechny klíče, dále pak všechny klíče z konkrétního modulu a nakonec všechny přímé klíče z konkretního modulu
Delete All Data	Test se pokusí smazat všechny uložené soubory včetně kořenového adresáře

Tabulka 6.5: Popis scénáře k unit testům.

6.6 Unit testy

Každý kvalitní software by měl být také dobře otestovaný. Proto je ke knihovně PISSD navržena sada unit testů, která má za úkol otestovat, zda všechny komponenty knihovny fungují jak mají. Testy se skládají z deseti testovacích scénářů. Některé z nich jsou rozdělené na několik podsekcí. Jednotlivé sekce a jejich popis lze nalézt v přiložené tabulce 6.5.

K vytvoření unit testů byla použita knihovna Catch2. Jedná se o testovací framework pro programovací jazyk C++, který je distribuovaný jako jediný hlavičkový soubor.

Kapitola 7

Rozhraní knihovny PISSD

V této kapitole bude popsáno aplikační rozhraní knihovny neboli API. Popisuje tedy veřejné metody určené pro použití uživatelem. Nenachází se zde ovšem detailní popis implementace, ani funkčnosti. V druhé půlce kapitoly jsou popsány klasické případy užití při několika typických scénářích použití knihovny.

7.1 SecureDataStorage

V této části bude popsána třída SecureDataStorage, která je jedinou třídou, s níž se uživatel setká. Slouží pro ukládání a opětovné získání uložených údajů. Dále také slouží ke kategorizaci dat a pohodlnou práci s moduly v nichž jsou data uložena. Kompletní seznam celého rozhraní a kratký popis lze nalézt v tabulce 7.1.

7.2 Práce s knihovnou PISSD

V této sekci budou uvedeny dva příklady typického použití knihovny za pomocí metod popsaných v této kapitole. První příklad ilustruje jednoduché uložení hesla uživatele a jeho opětovné získání a druhý příklad znázorňuje komplikovanější práci s moduly, jejich vytváření, vkládání dat do nich a jejich opětovné získání.

7.2.1 Jednoduché uložení hesla

První příklad popisuje získání hesla od uživatele, vytvoření instance třídy SecureDataStorage, následné uložení hesla a jeho opětovné získání.

Po získání uživatelského hesla ve formě stringu, je vytvořena instance třídy SecureDataStorage. Její konstruktor vyžaduje jako parametr ukazatel na třídu std::mutex. Následně je zavolána metoda storeData. Ta potřebuje dva parametry. Prvním je znakový řetězec sloužící jako klíč a druhým jsou samotná data. Metoda storeData je přetížena, takže není na uživatele vyvíjen tlak k zapamatovávání si mnoha metod. Ještě je vhodné zkontrolovat návratový kód, zda byl proces uložení úspěšný.

Pro opětovné získání uloženého hesla je zavolána metoda retrieveData. Ta má opět dva parametry. Prvním z nich je znakový řetězec sloužící jako klíč, jenž se musí shodovat s klíčem zadaným při ukládání dat. Druhým parametrem je proměnná ve formátu string, do níž bude v případě úspěchu heslo uloženo. Dále je vhodné zkontrolovat návratový kód metody retrieveData, aby bylo jisté, že operace proběhla v pořádku. Ukázku kódu lze nalézt ve výpisu 7.1.

Metoda	**Popis**
storeData	Metoda uloží data podle zadaného klíče
retrieveData	Metoda navrátí uložená data podle zadaného klíče
storeDataToModule	Metoda uloží data do zadaného modulu podle zadaného klíče
retrieveDataFromModule	Metoda navrátí uložená data ze zadaného modulu podle zadaného klíče
deleteStoredData	Metoda najde a vymaže požadovaná data podle zadaného klíče
deleteAllData	Metoda smaže všechna data a moduly včetně kořenového adresáře
deleteAllDataFromModule	Metoda vymaže všechna data v zadaném modulu
createModule	Metoda vytvoří požadovaný modul na zadané cestě
removeModule	Metoda ostraní požadovaný modul na zadané cestě
getAllKeys	Metoda navrátí klíče všech uložených dat ze všech modulů a cestu k nim
getAllKeysFromModule	Metoda navrátí všechny klíče ze zadaného modulu a jeho submodulů a cestu k nim
getDirectKeysFromModule	Metoda navratí všechny klíče ze zadaného modulu a cestu k nim
getAllModules	Metoda navrátí všechny moduly a cestu k nim
getAllSubmodules	Metoda navrátí všechny submoduly zadaného modulu
contains	Metoda vrátí informaci, zda hledaný klíč existuje v systému

Tabulka 7.1: Rozhraní třídy SecureDataStorage.

```
std::mutex MUTEX;

int storePassword(std::string &password)
{
    PISSD::SecureDataStorage storage(&MUTEX);
    if (storage.storeData("password", password) == 0)
        return 0;
    return -1;
}

int retrievePassword(std::string &password)
{
    PISSD::SecureDataStorage storage(&MUTEX);
    int returnCode = storage.retrieveData("password", password);
    if (returnCode == 0)
        return 0;
    else if (returnCode == 1)
        return 1;
    return -1;
}
```

Výpis 7.1: Ukázka uložení a načtení hesla pomocí knihovny PISSD.

7.2.2 Práce s daty v modulech

Druhý příklad popisuje vytvoření modulu, uložení několika dat do něj, jejich vyobrazení a následné získání konkrétního hesla zpět.

Po vytvoření instance třídy SecureDataStorage vytvoříme modul, do nějž chceme data ukládat pomocí metody createModule. Metoda createModule má dva parametry. První parametr je textový řetězec popisující cestu, kde má být modul vytvořen. V případě, že se jedná o kořenový adresář, tak jako první parametr se použije prázdný řetězec nebo hvězdička (*). Zanořené adresáře od sebe oddělujeme lomítkem (/). Druhý parametr je textový řetězec udávající název modulu. Po vytvoření modulu do něj ukládáme data pomocí metody storeDataToModule. Metoda má tři parametry, kdy prvním z nich je cesta k požadovanému modulu ve formě textového řetězce, zbylé dvě jsou stejné jako v případě metody storeData.

Před získáním uložených dat se nejprve pokusíme zjistit, zda námi hledaná data jsou vůbec v modulu uložena. K tomu použijeme funkci getDirectKeysFromModule, jenž má tři parametry. První specifikuje modul, ve kterém se má hledat. Ve zbylých dvou parametrech jsou navráceny nalezené klíče v modulu a cesty k nim. V případě, že modul obsahuje námi hledaný klíč, získáme jeho obsah pomocí metody retrieveDataFromModule. Ta se chová velice obdobně jako metoda retriveData, ale jako první parametr jí předkládáme cestu k námi požadovanému modulu. Dále je velice vhodné zkontrolovat návratový kód funkce retrieveDataFromModule, abychom měli jistotu, že vše proběhlo v pořádku a s daty je možné bezpečně pracovat. Příklad kódu lze nalézt ve výpisu 7.2.

```
std::mutex MUTEX;

void storePasswordstoModule(std::vector<std::string> passwords)
{
    PISSD::SecureDataStorage storage(&MUTEX);
    storage.createModule("*", "Passwords");
    int i = 0;
    for (auto pass : passwords)
    {
        storage.storeDataToModule("Passwords", std::to_string(i++), pass);
    }
}

int getOnePass(std::string dataKey, std::string &password)
{
    PISSD::SecureDataStorage storage(&MUTEX);
    std::vector<std::string> paths, dataKeys;
    storage.getDirectKeysFromModule("Passwords", paths, dataKeys);
    //kontrola existence by sla provest jednoduseji
    //postup slouzi jako priklad
    for (auto key : dataKeys)
    {
        if (key == dataKey)
        {
            if (0 == storage.retrieveDataFromModule("Passwords", key, password))
                return 0;
            return -1;
        }
    }
}
```

Výpis 7.2: Ukázka práce s moduly pomocí knihovy PISSD.

Kapitola 8

Testování

V této kapitole bude popsán experiment s časovou náročností a test funkčnosti redundance. Experiment si klade za úkol zjistit, zda knihovna PISSD nezatěžuje systém a využití knihovny PISSD výrazně nezpomaluje běh samotného programu, ve kterém je použita. Test má za úkol ověřit, že knihovna bude bezproblémově fungovat i při neopatrné manipulaci se soubory.

8.1 Experiment časové náročnosti

Cílem prvního experimentu bylo zjistit, jak moc je knihovna PISSD časově náročná v porovnání s uložením pouze nezašifrovaného textu. Smyslem bylo zjistit, zda je implementace dostatečně rychlá a její použití výrazně nezpomalí běh celé aplikace.

8.1.1 Zadání

Pro experimentování byly vytvořeny dva programy, jejichž cílem bylo vytvořit určitý počet souborů a zapsat do nich prostý text o délce 219 znaků. První program využíval knihovnu PISSD a její rozhraní, druhý program využíval standardní implementaci z knihovny fstream. Zdrojové kódy programu pro tvorbu souborů bez použití knihovny PISSD lze nalézt ve výpisu 8.1 a s použitím knihovny ve výpisu 8.2. Ve výpisu jsou použity tři druhy času: reálný, uživatelský a systémový. Reálný popisuje, jak dlouho se program vykonával. Uživatelský říká, kolik času procesor strávil během uživatelského kódu mimo jádro. A systémový čas popisuje, kolik času strávil procesor během vykonávání kódu v jádru.

Experimentování proběhlo ve třech fázích. Nejprve proběhl experiment s vytvořením 100 souborů, následně s 1000 a nakonec s 10 000 souborů. Všechny fáze proběhly 10x a jejich výsledek byl zprůměrován. Hodnoty byly naměřeny pomocí utility time dostupné na operačním systému macOS. Všechny hodnoty byly naměřeny na operačním systému macOS High Sierra ve verzi 10.13.4. Hardware byl postaven na MacBooku Pro, 2015 s dvou jádrovým procesorem Intel Core i5 taktovaném na 2,7 GHz s 8 GB pamětí DDR3, grafickou kartou Intel Iris Graphics 6100 s 1536 MB s primárním systémovým SSD diskem.

```
int main() {
    std::string data = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. "
                       "Nullam sit amet magna in magna gravida vehicula. Nam quis nulla.
```

```
                    "Integer vulputate sem a nibh rutrum consequat.";
    std::string dataKey = "dataKey";
    std::string test = "test/";

    for (int i = 0; i < 100; ++i)
    {
        dataKey = (std::to_string(i));
        dataKey = test + dataKey;
        std::ofstream outfile (dataKey);
        outfile << data;
        outfile.close();
    }
    return 0;
}
```

Výpis 8.1: Kód programu pro tvorbu souborů bez použití knihovny PISSD.

```
int main() {
    std::mutex myMutex;
    PISSD::SecureDataStorage storage(&myMutex);
    std::string data = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. "
                    "Nullam sit amet magna in magna gravida vehicula. Nam quis nulla.
                    "Integer vulputate sem a nibh rutrum consequat.";
    std::string dataKey = "dataKey";

    for (int i = 0; i < 100; ++i)
    {
        dataKey = std::to_string(i);
        storage.storeData(dataKey, data);
    }

    return 0;
}
```

Výpis 8.2: Kód programu pro tvorbu souborů s použitím knihovny PISSD.

8.1.2 Provedená zjištění

Z naměřených hodnot v tabulce 8.1 je patrné, že knihovna PISSD zpomaluje běh aplikace oproti běžnému nešifrovanému přístupu asi 14x. Základním předpokladem pro tak velký rozdíl naměřených hodnot je fakt, že knihovna PISSD vytváří 3x víc souborů. Samozřejmostí je potom i zvýšená režie na zašifrováni textu, hledaní cest a velké množství bezpečnostních kontrol. Na obrázku 8.1 lze nalézt graf popisující, kolikrát je doba běhu aplikace s knihovnou PISSD delší než se standardní implementací z knihovny fstream. Jak můžeme vidět, počet souborů nemá výrazný vliv na dobu trvání jedné operace.

Počet souborů	Použití PISSD	Typ času	Celkový čas/s	Čas na 1 operaci/ms
100	NE	real	0.029	0.29
100	NE	user	0.003	0.03
100	NE	sys	0.025	0.25
100	ANO	real	0.318	3.18
100	ANO	user	0.269	2.69
100	ANO	sys	0.047	0.47
1000	NE	real	0.206	0.206
1000	NE	user	0.013	0.013
1000	NE	sys	0.173	0.173
1000	ANO	real	3.155	3.155
1000	ANO	user	2.679	2.679
1000	ANO	sys	0.459	0.459
10 000	NE	real	2.225	0.2225
10 000	NE	user	0.137	0.0137
10 000	NE	sys	0.1879	0.01879
10 000	ANO	real	32.697	3.2697
10 000	ANO	user	26.867	2.6867
10 000	ANO	sys	5.035	0.5035

Tabulka 8.1: Výsledky rychlostního experimentu.

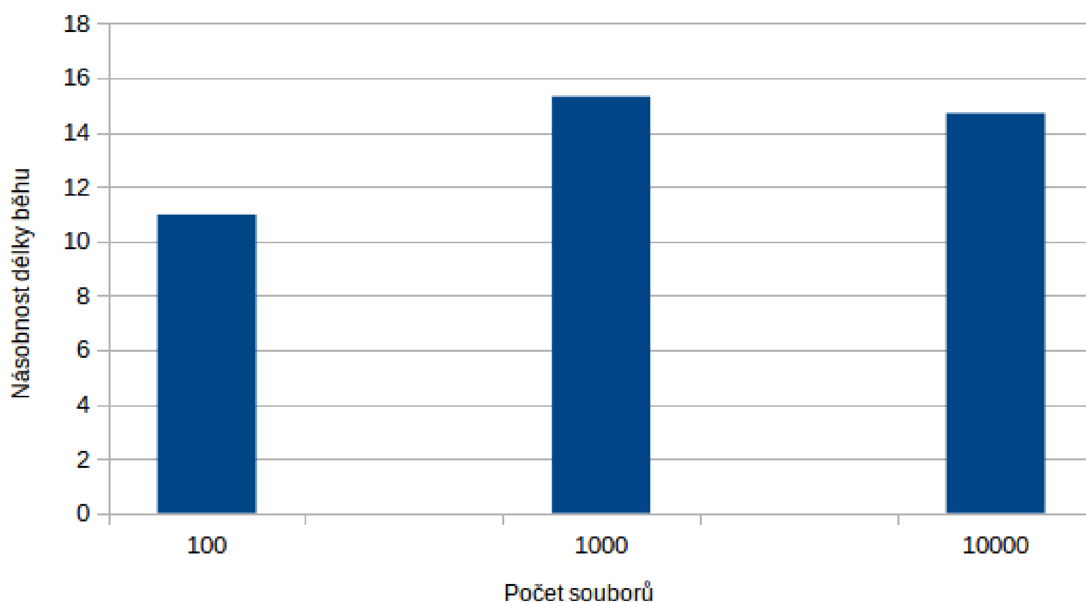
Jak je z provedeného experimentu patrné, knihovna PISSD by měla být používána pouze na šifrování malého počtu souborů, které jsou opravdu nezbytné pro běh programu. Každopádně při použití knihovny jak byla zamýšlena, je zpoždění zanedbatelné a nebude mít vliv na běh programu.

8.2 Test funkčnosti redundance

Cílem druhého testu bylo zjistit, jestli je knihovna PISSD odolná vůči poškození a ztrátě dat. Hlavní snahou bylo prokázat, že knihovna v případě poškození jednoho či dvou souborů bude schopna získat ztracená data zpět a informovat uživatele o této zkušenosti.

8.2.1 Zadání

Pomocí knihovny PISSD byl vytvořen a zašifrován jeden soubor se kterým bylo provedeno sedm testovacích scénářů. V prvním případě byl jeden soubor změněn. V druhém testu byl jeden soubor smazán. Ve třetím scenáří byl jeden soubor smazán a druhý pozměněn. Ve čtvrtém zadání byly dva soubory pozměněny. V pátem případě byly dva soubory smazány. V šestém testovacím scénáři byly dva soubory smazány a třetí byl upraven. A v posledním sedmém případě byly všechny tři soubory smazány. Všechny testy byly provedeny nezávisle na sobě. Při testovaní byl pozorován návratový kód metody retrieveData a obsah získaných dat pomocí této metody.



Obrázek 8.1: Graf zobrazení násobnosti délky běhu programu oproti implementaci s fsream.

Popis	**Navrácená data**	**Návratový kód**
1 změněn	Původní	1
1 smazán	Původní	1
1 smazán + 1 změněn	Původní	1
2 změněny	Původní	1
2 smazány	Původní	1
2 smazány + 1 změněn	Prázdná	-1
3 smazány	Prázdná	-1

Tabulka 8.2: Výsledky bezpečnostního testu.

8.2.2 Provedená zjištění

Podle dat uvedených v tabulce 8.2 se prokázalo, že knihovna PISSD pracuje tak, jak byla zamýšlena. Ve všech případech, kdy byl minimálně jeden soubor v pořádku, byl navrácen správný obsah souboru a návratový kód 1. V posledních dvou případech, kdy byly smazány nebo pozměněny všechny soubory s uloženými daty, byl návratový kód -1 a proměnná s daty byla prázdná.

Díky provedeným testům si může být uživatel knihovny jistý, že pokud dojde k poškození nebo ztrátě uložených dat bude uživatel informován o této skutečnosti. Tento přístup by mu měl zajistit bezpečnou práci se získanými daty i v případě, že neproběhne vše bez chyby.

Kapitola 9

Závěr

Cílem práce bylo seznámit se s možnostmi šifrování dat, bezpečnému přístupu k nim a prozkoumat existující řešení nástrojů pro ukládání citlivých dat. Na základě získaných znalostí byla navržena a implementována knihovna PISSD. Jako důkaz funkčnosti vznikla v rámci knihovny sada unit testů a několik experimentů.

Bylo zjištěno, že neexistuje žádný podobný volně dostupný nástroj, jenž by vyhovoval požadavkům této práce. Proto jsem v návrhu knihovny nemohl vycházet z žádných dostupných prostředků a bylo potřeba navrhnout co možná nejlepší řešení pouze na základě znalostí šifrovacích principů a ověřených metod.

Hlavní podmínkou v zadání bylo především jednoduché aplikační rozhraní, což se promítlo do celého návrhu. Aby uživatel nemusel zadávat heslo, vznikl komplikovaný systém generace hesla. To je generováno způsobem, aby bylo unikátní pro každého uživatele na každém zařízení. Tento přístup jej odlišuje od většiny ostatních kryptografických softwarů.

Knihovna kromě ukládání a získávání dat v různých formátech, také umožňuje strukturování těchto dat do modulů, jejich vytváření, mazání a několik doprovodných metod pro pohodlnou práci s ní. Další velikou předností knihovny je její bezpečnost při práci s více vlákny. Knihovna také zajišťuje ochranu před náhodným smazáním tím, že zálohuje uložená data na třech místech zároveň.

S knihovnou bylo provedeno několik experimentů, které měly za úkol dokázat její nenáročnost na výkon aplikace a její bezpečnost. Bylo zjištěno, že ukládání pomocí knihovny PISSD je 14x pomalejší oproti standardnímu ukládání. Avšak domnívám se, že zvýšená režie na vytvoření záloh a zašifrování je více než přiměřeně velká bezpečnosti jenž přináší. Také tímto experimentem bylo dokázáno, že počtem souborů není výrazně ovlivněna délka jedné operace. V dalším testu bylo prokázáno, že při smazání nebo úpravě jednoho či dvou souborů je knihovna schopná zajistit původní uložený obsah, a navíc o něm uživatele informovat, tak aby věděl, že s daty bylo manipulováno. V případě smazání všech tří záloh jsou data nenávratně ztracena a již s nimi nelze více pracovat.

Možnosti dalšího rozvoje této práce jsou ve vylepšení knihovny v oblasti implementace samotného zabezpečení a rychlosti ukládání a opětovného získávání dat. V experimentech bylo prokázáno, že knihovna má dopad na rychlost běhu programu, a proto má v tomto ohledu ještě velký potenciál pro zlepšení. Samozřejmostí je také zlepšovat kryptografické algoritmy a techniky, neboť nároky na zabezpečení se zvětšují každý den. Na druhou stranu se domnívám, že nemá smysl rozšiřovat aplikační rozhraní, protože by tím byla ztracena původní myšlenka.

Literatura

[1] *Crypto++ 6.0.0 Benchmarks*. [Online; navštíveno 20.02.2018].
URL https://www.cryptopp.com/benchmarks.html

[2] Apple: *iOS Security iOS 11*. [Online; navštíveno 15.02.2018].
URL https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[3] Bertoni, G.; Daemen, J.; Peeters, M.; aj.: Sponge Functions. 2007.
URL https://keccak.team/files/SpongeFunctions.pdf

[4] Bogdanov, A.; Khovratovich, D.; Rechberger, C.: Biclique Cryptanalysis of the Full AES. In *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, Berlin, Heidelberg: Springer-Verlag, 2011, ISBN 978-3-642-25384-3, s. 344–371, doi:10.1007/978-3-642-25385-0_19.
URL http://dx.doi.org/10.1007/978-3-642-25385-0_19

[5] Buchmann, J. A.: *Introduction to Cryptography*. Springer Verlag, 2002, ISBN 0-387-95034-6.

[6] Calderbank, M.: The RSA Cryptosystem: History, Algorithm, Primes. 2007.

[7] Daemen, J.; Rijmen, V.: AES Proposal: Rijndael. 1999.

[8] Doseděl, T.: *Počítačová bezpečnost a ochrana dat*. Computer Press, 2004, ISBN 80-251-0106-1.

[9] Dworkin, M. J.: SP 800-38A 2001 Edition. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Technická zpráva, Gaithersburg, MD, United States, 2001.

[10] Elbirt, A. J.: *Understanding and applying cryptography and data*. CRC Press, 2009, ISBN 978-1-4200-6160-4.

[11] Huseby, S. H.: *Zranitelný kód*. Computer Press, 2006, ISBN 80-251-1180-6.

[12] Çetin K. Koç; Naccache, D.; Paar, C.: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Springer, 2001, ISBN 978-3-540-42521-2.

[13] Kahn, D.: *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996, ISBN 0-684-83130-9.

[14] Kaliski, B. S.: *Advances in Cryptology-CRYPTO97*. Springer Verlag, 1997, ISBN 3-540-63384-7.

[15] Katz, J.; Lindell, Y.: *Introduction to Modern Cryptography: Principles and Protocols*. Chapman and Hall/CRC, 2007, ISBN 1584885513.

[16] Konheim, A. G.: *Hashing in computer science : fifty years of slicing and dicing*. Wiley, 2010, ISBN 978-0-470-34473-6.

[17] Krawczyk, H.: *Advances in Cryptology - CRYPTO '98*. Springer, 1998, ISBN 978-3-540-64892-5.

[18] Luby, M.: *Pseudorandomness and cryptographic applications*. Princeton University Press, 1996, ISBN 978-0-691-02546-9.

[19] Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A.: *Handbook of applied cryptography*. CRC Press, 1996, ISBN 0-8493-8523-7.

[20] Paar, C.; Pelzl, J.: *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer, 2010, ISBN 3-642-04100-0.

[21] Pokorný, J.: *Hacking : umění exploitace*. Zoner Press, 2009, ISBN 978-80-7413-022-9.

[22] Reynard, R.: *Secret Code Breaker II: A Cryptanalyst's Handbook*. Smith & Daniel Marketing, 1997, ISBN 1-889668-06-0.

[23] Russel, F.: *Information Gathering in Classical Greece*. Michigan Press, 1999, ISBN 0-472-11064-0.

[24] Salah, I.; Darwish, A.; Oqeili, S.: Mathematical Attacks on RSA Cryptosystem. ročník 2, 08 2006.

[25] Schmeh, K.: *Cryptography and public key infrastructure on the Internet*. John Wiley & Sons, 2003, ISBN 978-0-470-84745-9.

[26] Shanon, C. E.: *Communication Theory of Secrecy Systems*. *Bell System Technical Journal*, ročník 27, č. 4, 1949: s. 656–715, ISSN 0005-8580.

[27] Singh, S.: *Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii*. Dokořán, 2003, ISBN 80-86569-18-7.

[28] van Tilborg, H. C.; Jajodia, S.: *Encyclopedia of Cryptography and Security*. Springer, 2011, ISBN 978-1-4419-5907-2.

Příloha A

Instalace

V této kapitole bude popsána instalace knihovny PISSD a všech jejich prerekvizit. Zdrojové kódy této práce jsou dostupné z repozitáře verzovacího systému GitHub. Zdrojové kódy lze stáhnout přímo z webové stránky nebo pomocí terminálového příkazu:

$$git\ clone\ https://github.com/Tasarak/libPISSD$$

Před samotným překladem knihovny, je nejprve potřeba nainstalovat knihovny Crypto++ a Boost. Na operačním systému MacOs lze knihovnu Crypto++ stáhnout a nainstalovat pomocí nástroje Brew pomocí terminálového příkazu:

$$brew\ install\ Crypto++$$

Dále je potřeba nainstalovat knihovnu Boost. Na operačním systému MacOS lze knihovnu opět stáhnout a nainstalovat pomocí nástroje Brew pomocí terminálového příkazu:

$$brew\ install\ boost$$

Ve chvíli, kdy jsou splněny všechny závíslosti, je možné sestavit knihovnu pomocí programu CMake. Po něm je nutné použit program make, který přeloží knihovnu a zároveň vytvoří spustitelný soubor PISSD_unit_tests, obsahující sadu unit testů. Je vhodné ověřit funkčnost knihovny před začátkem jejího používání spuštěním unit testů.

Pro práci s knihovnou pod operačním systémem Windows je potřeba nejprve zkompilovat knihovnu Crypto++ a Boost a přidat podporu pro tvar cest k souborům používaných v operačních systémech Windows. Kompilace knihovny by měla proběhnout pomocí stejných nástrojů a postupů.

Příloha B

Obsah CD

- xkleme11.pdf - elektronická verze textu této práce
- xkleme11.zip - zdrojové soubory textu této práce
- sources.zip - zdrojové soubory implementované knihovny PISSD