VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ÚTOK NA ŠIFROVÁNÍ DOKUMENTŮ OPENDOCUMENT S VYUŽITÍM GPU

ATTACK ON OPENDOCUMENT ENCRYPTION USING GPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **SAMUEL JAKUBÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK HRANICKÝ**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19884/2016/xjakub31

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Jakubík Samuel**
Obor: Informační technologie
Téma: **Útok na šifrování dokumentů OpenDocument s využitím GPU**
Attack On OpenDocument Encryption Using GPU
Kategorie: Bezpečnost

Pokyny:

1. Seznamte se s architekturou a implementací nástroje Fitcrack/Wrathion.
2. Nastudujte techniky šifrování dokumentů OpenDocument.
3. Navrhněte rozšíření modulů programu z bodu 1, které doplní jeho funkcionalitu o podporu formátů OpenDocument.
4. Navržené rozšíření implementujte (včetně kódu pro akceleraci pomocí GPU).
5. Proveďte měření výkonu navržených modulů a výsledky porovnejte s konkurenčními nástroji.
6. Zhodnoťte dosažené výsledky.

Literatura:

- WEIR J., OpenDocument Format: The Standard for Office Documents. *IEEE Internet Computing*, roč. 13, č. 2, s. 83-87. IEEE 2009. ISSN 1089-7801.
- MENEZES, Alfred J., VAN OORSCHOT, Paul C., VANSTONE, Scott A. Handbook of Applied Cryptography. Boca Raton (Florida): CRC Press 1999. 810 s. ISBN 978-8189836122.
- A další dle dohody s vedoucím.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

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: **Hranický Radek, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cieľom práce je rozšíriť nástroj Fitcrack o modul umožňujúci obnovu hesiel z dokumentov formátu Open Document Format, pracujúci na CPU, ale taktiež využívajúci podporu GPU na jeho zrýchlenie. V práci je analyzovaný formát súborov Open Document Format a následne sú v nej rozobrané funkcie a algoritmy, ktoré sú využívané na ich zabezpečenie. Na implementáciu modulu pre GPU je využitý framework OpenCL. Pre výsledný modul bol nameraný čas, potrebný na obnovu hesla a zrýchlenie tohto procesu pri využití grafickej karty. Práca obsahuje aj porovnanie s existujúcim nástrojom, ktorý podporuje dešifrovanie tohto formátu.

Abstract

The aim of this work is to extend Fitcrack with a module that allows the recovery of passwords from Open Document Format documents using either CPU or GPU to accelerate the process. The thesis further analyses the structure of the Open Document Format. Subsequently, it describes the functions and algorithms used to encrypt Open Document Format files. The GPU module is implemented with the use of the OpenCL framework. Both the time needed to recover the password and the acceleration of the process using the graphics card were measured. The work also contains a comparison between an existing tool that supports decryption of this format and the newly implemented module.

Kľúčové slová

kryptografia, Fitcrack, obnova hesiel, Open Document Format, OpenCL

Keywords

cryptography, Fitcrack, password recovery, Open Document Format, OpenCL

Citácia

JAKUBÍK, Samuel. *Útok na šifrování dokumentů OpenDocument s využitím GPU*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický

Útok na šifrování dokumentů OpenDocument s využitím GPU

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne, pod vedením Ing. Radka Hranického.

.....
Samuel Jakubík
15. mája 2017

Poďakovanie

Rád by som touto cestou poďakoval Ing. Radkovi Hranickému za odborné vedenie, ústretovosť pri konzultáciách, ochotu pomôcť a cenné rady pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Už od pradávnych čias sa ľudstvo snažilo nájsť rôzne spôsoby predávania si informácií. Od nástenných malieb či dymových signálov sme dospeli až k písaným listom a elektronickým dokumentom. S postupným skvalitňovaním prenosu informácií však prišla otázka ich zabezpečenia v prípade odcudzenia. Či už to boli príkazy pre armádu, ktoré zachytila nepriateľská rozviedka, alebo dôležitý dokument nejakej firmy, prípadne vlády, ktorý sa dostal do nepoverených rúk, začala táto otázka zabezpečenia naberať na vážnosti. Od Cézarovej šifry, cez Enigmu až po dnešné šifry ako Blowfish alebo AES, sa začali správy šifrovať a tak ochraňovať ich obsah aj v prípade ich odcudzenia.

S týmto novým zabezpečením sa však vyskytli nové problémy. Pokiaľ človek, ktorý pôvodnú správu alebo dokument zašifroval, zabudne heslo, nie je možné bez jeho pomoci obsah spätne rozlúštiť. Ďalším problémom je verejná dostupnosť nástrojov, ktoré umožňujú zašifrovať informácie ktorémukoľvek užívateľovi, bez ohľadu na dôvod i obsah súboru. Toto môže byť nežiaduce v tom prípade, keď ide o osobu, ktorá spáchala nejaký trestný čin a obsahom dokumentu sú potenciálne dôkazy, ktoré by mohli viesť k jej usvedčeniu.

Postupným rozširovaním počítačových technológií vo firmách a verejnej správe sa začali dokumenty uchovávať nielen v papierovej, ale aj v elektronickej forme. Každá firma však mohla vytvárať tieto dokumenty v rozdielnych programoch, ktoré nemuseli poskytovať kompatibilitu výsledných súborov s ostatnými programami. Z tohto dôvodu vznikol Open Document Format (ODF). Ako už bolo spomenuté, kybernetická kriminalita, strata hesiel a tým aj všetkých dát, sú problémami, ktoré je potrebné riešiť. Obnova hesla za čo najkratší čas je hlavne pri vyšetrovaní zločinu veľmi dôležitá. Za týmto účelom boli vyvinuté nástroje ako je Fitcrack, ktoré sa snažia urýchliť proces obnovy hesiel za pomoci podpory grafických kariet. Nástroj Fitcrack však vo svojej funkcionalite zatiaľ neobsahuje podporu pre obnovu hesiel zo súborov Open Document Format. Práca je preto zameraná na rozšírenie o implementáciu modulu práve o tento formát súborov. V práci sú ďalej rozobraté niektoré z funkcií, ktoré sú používané na zabezpečenie súborov ODF.

V kapitole 2 je bližšie popísaný princíp fungovania programu Fitcrack a možnosti jeho rozšíriteľnosti. Kapitola 3 vysvetľuje štandard OpenCL, ktorého využitím bude sprístupnená akcelerácia na GPU. V kapitolách 4 a 5 sú rozobrané kryptografické hašovacie funkcie a šifrovacie algoritmy. Obsahom kapitoly 6 je popis štruktúry súborov formátu OpenDocument Format (ODF). Návrh modulu, ktorý rozširuje nástroj Fitcrack, sa nachádza v kapitole 7. Implementácia modulu je obsahom kapitoly 8 a kapitola 9 zahŕňa experimenty a merania s už implementovaným modulom ODF.

Kapitola 2

Fitcrack

V tejto kapitole je popísaný nástroj Fitcrack, ktorý vznikol ako nástupca nástroja Wrathion. Wrathion vytvoril Ing. Jan Schmied v roku 2014 v rámci jeho diplomovej práce [11]. Išlo o nástroj na obnovu hesiel súborov formátu PDF, DOC a ZIP. Pretože je to výpočetne náročná práca a samotné CPU pri dlhších a zložitejších heslách nemusí byť dostatočne rýchle, obidva nástroje podporujú akceleráciu na GPU pomocou OpenCL a CUDA. Nástroj Fitcrack je ďalej rozšírený o nové moduly (napr. RAR, 7z) a možnosť distribuovaných výpočtov s využitím platformy BOINC[1]. Na obrázku 2.3 sú v tabuľke zobrazené všetky formáty súborov a typy šifrovania, ktoré momentálne nástroj Fitcrack podporuje.

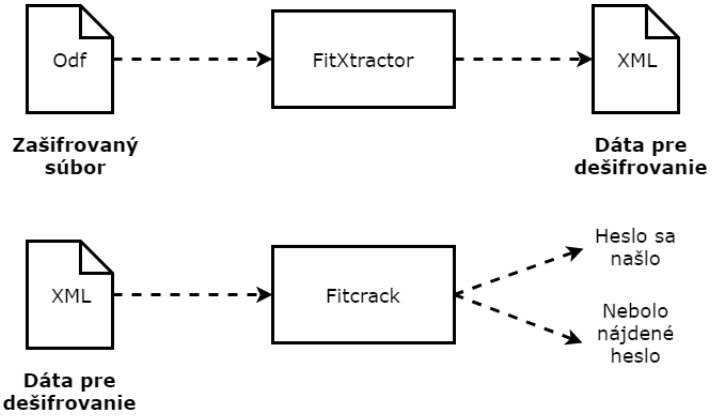
2.1 Architektúra

Nástroj je rozdelený do troch častí pre jednoduchšiu škálovateľnosť. Toto rozdelenie je znázornené na obrázku 2.2. Týmito časťami sú:

- **Generator** — obsahuje funkcionalitu potrebnú pre samotné generovanie hesiel.
- **Cracker** — časť obsahujúca kódy, pre CPU aj GPU, na obnovu hesiel zo súborov rôznych formátov.
- **Controller** — ovláda proces obnovy hesiel. Spúšťa generovanie hesiel, Cracker a komunikuje s BOINC klientom pri distributívnom riešení.

Pretože niektoré druhy zabezpečenia môžu vyžadovať časť alebo aj celý obsah súboru na dešifráciu, bol nástroj Fitcrack, oproti Wrathionu, rozdelený na dve časti. Týmito časťami sú teda dva samostatne spustiteľné programy Fitcrack a FitXtractor. Program FitXtractor zistí formát zadaného súboru a na základe toho zvolí modul, ktorý sa použije. Zvolený modul potom skontroluje, či zadaný súbor je zašifrovaný a pokiaľ tomu tak je, zistí zo súboru potrebné informácie na dešifrovanie a overenie hesla. Tieto informácie potom uloží v XML formáte do súboru. Ten má typicky, oproti pôvodnému súboru, oveľa menšiu veľkosť, no nemusí to platiť vždy. Pre Fitcrack, ktorý je zodpovedný za samotnú obnovu hesiel, je tento spôsob extrahovania dát do menšieho súboru dôležitý. Najmä z hľadiska distribuovaného riešenia, pri ktorom nie je zrovna najlepším nápadom posielať niekoľko megabajtov dát po sieti. Tento postup je znázornený na obrázku 2.1

[1] Berkeley Open Infrastructure for Network Computing (BOINC) – `http://boinc.berkeley.edu/`

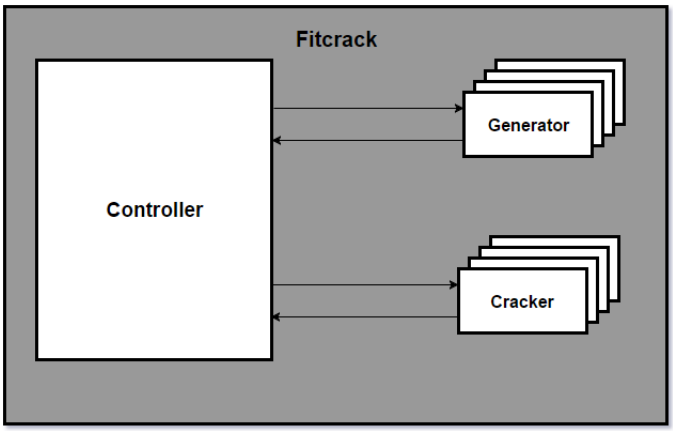


Obr. 2.1: Postup dešifrácie dokumentu využitím nástrojov FitXtractor a Fitcrack

2.2 Generovanie hesiel

Generovanie hesiel je nevyhnutnou časťou každého programu na obnovu hesiel. Nástroj na generovanie hesiel podporuje aj znaky nevyskytujúce sa v ASCII tabuľke. V súčasnosti Fitcrack využíva na generovanie hesiel jeden z troch princípov:

- **Brute-force** — vytváranie všetkých možných reťazcov nad zadanou znakovou sadou.
- **Rule-based** — podobné ako brute-force, ale navyše je možné zadať rôzne doplňujúce pravidlá (napr. že prvý znak je číslo, za ktorým musí nasledovať veľké písmeno, alebo ďalšie číslo).
- **Dictionary** — tento prístup využíva slovník ako zdroj možných hesiel.



Obr. 2.2: Architektúra nástroja Fitcrack [8]

2.3 Overenie hesla

Overovanie hesiel prebieha rozdielne v závislosti na formáte a použitom šifrovaní súboru. Modul porovnáva výsledky výpočtov hašovacej a šifrovacej funkcie s kontrolnou hodnotou, ktorú zo súboru vyextrahoval FitXtractor.

2.3.1 CPU

Overovanie na CPU má jednoduchý priebeh:

- vygeneruje sa heslo,
- overí sa správnosť vygenerovaného hesla,
- pokiaľ je vygenerované heslo správne, program sa ukončí,
- v opačnom prípade sa vygeneruje ďalšie heslo a pokračuje sa v procese overovania.

2.3.2 GPU

Pri overovaní s GPU je potreba inicializovať systém OpenCL, načítať a preložiť program. Po týchto krokoch môžeme do GPU nahrať dáta. Keďže OpenCL zakazuje na GPU možnosť dynamickej realokácie, je nutné alokovať pamäť dostatočne veľkú pre celý beh aplikácie. Celý proces môže potom bežať v jednom z dvoch režimov podľa toho, kde sú heslá generované:

- **CPU** — brute-force/dictionary, je potrebné stále posielať heslá medzi pamäťami;
- **GPU** — len brute-force, heslá sú generované priamo do privátnej pamäte GPU.

Format	Version	Encryption	Verification	CPU	OpenCL	CUDA
PDF	1.1 - 1.4 (Acrobat 5)	RC4 40-bit	MD5, RC4	YES	YES	YES
	1.4 (Acrobat 5)	RC4 128-bit	MD5, RC4	YES	YES	YES
	1.6 (Acrobat 6)	AES 128-bit	MD5, RC4	YES	YES	YES
	1.7 Extension Level 3 (Acrobat 9)	AES 256-bit	SHA-256	YES	YES	YES
	1.7 Extension Level 8 - 2.0	AES 256-bit	SHA-256, SHA-384, SHA-512, AES-128-CBC	in development		
ZIP	ZIP 2.0 Legacy (PKZIP)	PKZIP stream cipher	PKZIP stream cipher, CRC-32	YES	YES	YES
	Strong encryption (WinZIP)	AES 128/192/256-bit	AES 128/192/256-bit, PBKDF2, SHA1	YES	YES	YES
	ZIP 5.2+ (SecureZIP) with AES	AES 128/192/256-bit	AES 128/192/256-bit, CRC-32	YES	YES	YES
7z	7-Zip	7zAES 256-bit	7zAES 256-bit, SHA-256, LZMA for decompression	YES	YES	NO
RAR	Version 3	AES-128-CBC	SHA1, CRC	YES	YES	NO
	Version 3 - uncompressed	AES-128-CBC	SHA1, CRC	YES	YES	NO
	Version 3 - encrypted header	AES-128-CBC	SHA1	YES	YES	NO
	Version 5	AES-256-CBC	BKPDF2-HMAC-SHA-256	YES	YES	NO
DOC, XLS	97-2000	RC4 40-bit	MD5, RC4	YES	YES	YES
	XP	RC4 128-bit	SHA1, RC4	YES	YES	YES
	2003	RC4 128-bit	SHA1, RC4	YES	YES	YES
PPT	XP	RC4 128-bit	SHA1, CRC	YES	YES	YES
	2003	RC4 128-bit	SHA1, CRC	YES	YES	YES
DOCX, XLS, PPTX	2007	AES 128/192/256-bit	SHA1, AES	YES	YES	YES
	2010	AES 128/192/256-bit	SHA1, SHA256, SHA512, AES	YES	YES	YES
	2013	AES 128/192/256-bit	SHA1, SHA256, SHA512, AES	YES	YES	YES
	2016	AES 128/192/256-bit	SHA1, SHA256, SHA512, AES	YES	YES	YES

Obr. 2.3: Aktuálne podporované formáty dokumentov a šifrovacie algoritmy nástrojom Fitcrack[3]

[3]Zdroj: `http://wrathion.fit.vutbr.cz/fitcrack/?i=features`

Kapitola 3

OpenCL

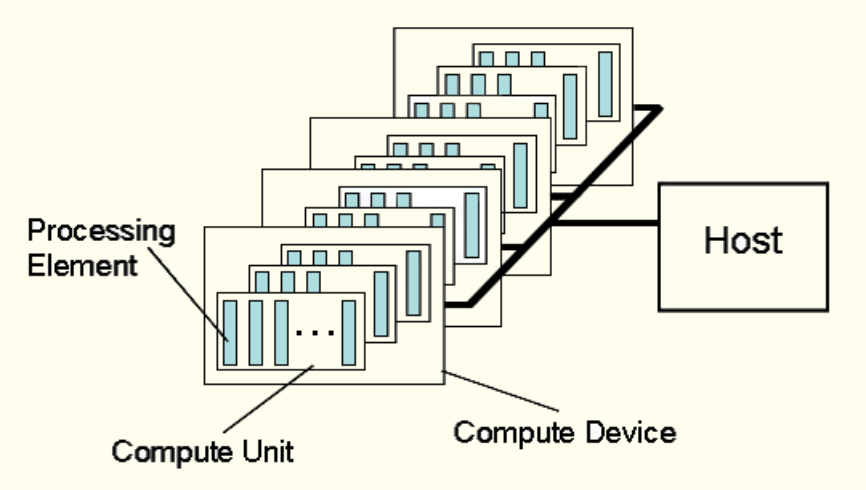
Open Computing Language (OpenCL) je otvorený nespoplatnený priemyslový štandard, umožňujúci vývojárom paralelné programovanie a využitie potenciálu výpočtových jednotiek ako sú CPU a GPU. Za vznikom a vývojom OpenCL stojí konzorcium Khronos, ktoré združuje firmy ako AMD, NVIDIA a mnohé iné. Je nezávislý na hardwarovej či softwarovej platforme a vďaka práci na nízkej úrovni poskytuje vysokú efektivitu [7]. Programovanie samotnej aplikácie sa delí na:

- Programovanie **hostiteľskej aplikácie**, ktorá je vykonávaná na strane riadiaceho procesoru. K tomu štandard poskytuje rozhranie pre jazyky C a C++, no vďaka rozhraniam tretích strán je možné použiť aj Python, Java alebo .NET.
- Programovanie **OpenCL zariadenia**, teda vytváranie funkcií pre grafickú kartu. Táto funkcia je nazývaná aj **kernel**. Na ich programovanie slúžia jazyky OpenCL C a od verzie 2.1 aj OpenCL C++. Jazyky sú podmnožinami C99 (OpenCL C) a C++14 (OpenCL C++), rozšírenými o rôzne funkcie a dátové štruktúry, napríklad *vektor*. Majú však aj vlastné obmedzenia ako napríklad zakázané rekurzívne volanie funkcií alebo polia premenlivej dĺžky.

Architektúra OpenCL je opísateľná pomocou hierarchie štyroch nasledujúcich modelov:

- Model platformy – opisuje vzťah medzi zariadeniami OpenCL a hostiteľskou aplikáciou.
- Pamäťový model – definuje štruktúru a správanie pamäte počas behu OpenCL programu.
- Exekučný model – určuje abstraktné rozloženie OpenCL funkcií (kernelov) na zariadení.
- Programový model – definuje spôsob paralelného programovania.

Tieto modely sú abstraktným vyjadrením správania a vlastností platforiem a zariadení, ktoré zodpovedajú štandardu OpenCL.



Obr. 3.1: Model platformy [7] — hostiteľ (host), výpočtové zariadenie (compute device), výpočtová jednotka (compute unit), procesný prvok (processing element)

3.1 Model platformy

Definuje prácu na najvyššej úrovni. Model pozostáva z jedného hostiteľa (obvykle CPU) pripojeného na jedno alebo viacero OpenCL zariadení (napr. GPU). Tieto zariadenia sú ďalej rozdelené do jednej či viacerých výpočtových jednotiek (compute units), ktoré sa skladajú z procesných prvkov (processing elements). Tie vykonávajú výpočet na danom zariadení.

3.2 Exekučný model

Program je vykonávaný v dvoch úrovniach – v hostiteľskom programe a v programe pre zariadenia, ktorý je tvorený jedným alebo viacerými kernelmi (funkciami v OpenCL C). Aplikačná časť je vykonávaná hostiteľom (CPU). Zodpovedá za komunikáciu medzi hostiteľom a zariadeniami (GPU) a taktiež za spustenie a koordináciu výpočtu na zariadeniach. O samotné výpočty sa stará kernel.

Výpočty prebiehajú v pracovných jednotkách (work-item), ktoré majú svoje jednoznačné identifikátory. Tieto jednotky sú spúšťané v pracovných skupinách (work-group), ktoré majú taktiež svoje identifikátory. Každá pracovná jednotka má aj identifikátor v rámci svojej pracovnej skupiny.

Aplikačná časť je ďalej zodpovedná za vytvorenie takzvaného kontextu. Ten obsahuje:

- informácie o OpenCL zariadeniach, ktoré hosť bude využívať,
- kernely (OpenCL funkcie), ktoré budú zariadenia spúšťať,
- program, ktorý implementuje dané kernely,
- skupinu objektov v pamäti, ktoré sú viditeľné pre OpenCL zariadenia a obsahujú dáta, s ktorými môžu kernely pracovať.

Pre ovládanie a komunikáciu so zariadeniami využíva hostiteľ fronty príkazov. Táto fronta je vytvorená hostiteľom a priradená jednému OpenCL zariadeniu, po vytvorení kontextu. Príkazy vo fronte môžu byť spracovávané [10]:

- v tom poradí v akom do fronty prišli (*In-order*).
- nezávisle na poradí v akom do fronty prišli, ale v závislosti na explicitne definovaných synchronizačných bodoch, alebo explicitne zadaných závislostiach na udalostiach (*Out-of-order*).

3.3 Pamäťový model

Tento model opisuje štruktúru, obsah a správanie pamäte OpenCL platformy počas behu programu. Ďalej špecifikuje typy pamäte, druhy prístupu a jej alokáciu. Pamäťový model definujeme v štyroch častiach:

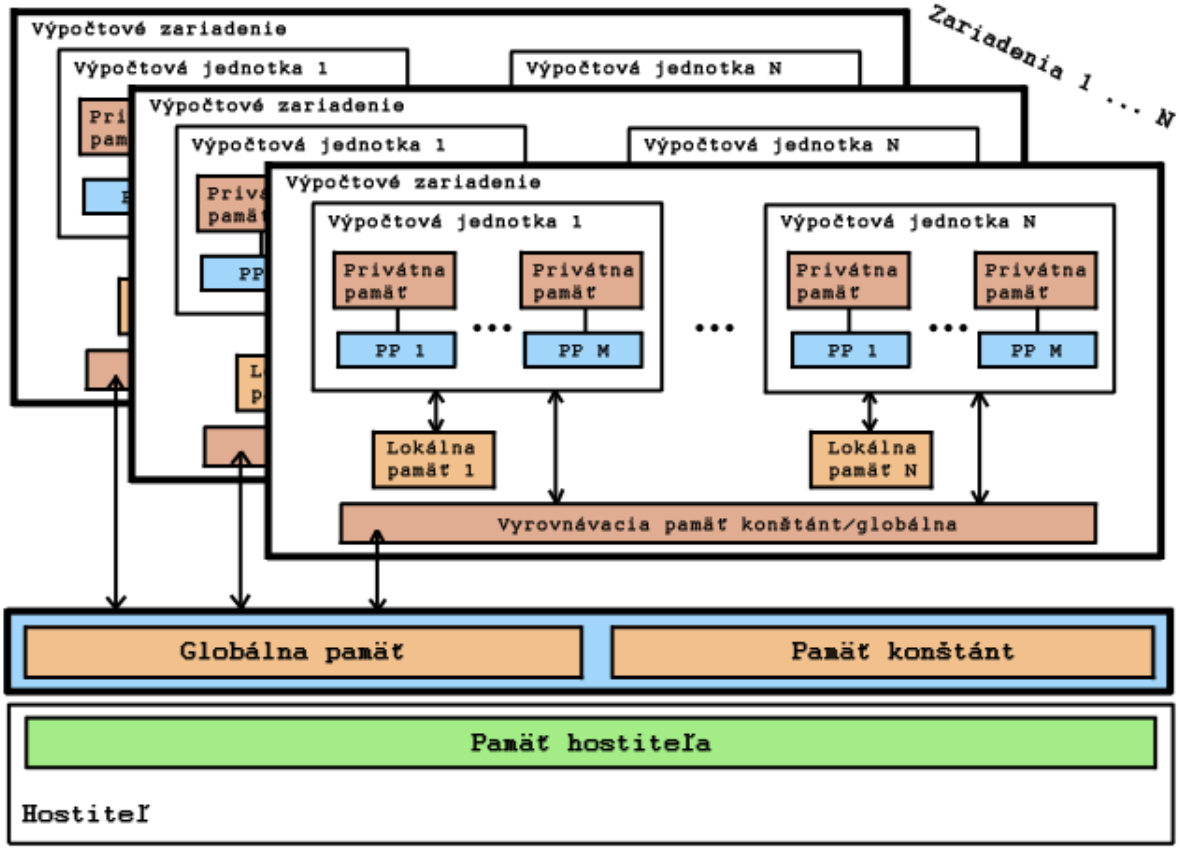
- Pamäťové oblasti — definujú jednotlivé časti pamäte, ktoré sú dostupné hostiteľovi a zariadeniu v rovnakom kontexte.
- Pamäťové objekty — definované v OpenCL API, spravované hostiteľom a zariadeniami.
- Zdieľaná virtuálna pamäť — virtuálny adresový priestor prístupný hostiteľovi a zariadeniam v rovnakom kontexte.
- Model konzistencie — pravidlá pre oblasť pamäte, ktoré sú používané viacerými jednotkami súčasne a zaručujú, že dáta ostávajú validné.

Pamäť v OpenCL je rozdelená na dve časti -– pamäť hostiteľa a pamäť zariadenia. Pamäť hostiteľa je priamo dostupná iba hostiteľovi a jej správanie je definované mimo OpenCL. Pamäť zariadenia je dostupná kernelom bežiacim na zariadeniach. Táto časť pozostáva zo štyroch pamäťových priestorov:

- Globálna pamäť – táto oblasť pamäte je dostupná na čítanie aj zápis všetkým pracovným jednotkám vo všetkých pracovných skupinách, výmenou za dlhšiu prístupovú dobu.
- Pamäť konštánt – je región globálnej pamäte, ktorý je konštantný počas doby behu kernelu. Pracovné jednotky môžu k dátam tejto pamäte pristupovať, ale nemôžu ich žiadnym spôsobom meniť. Hostiteľ alokuje a inicializuje pamäťové objekty, umiestňované do tejto pamäte.
- Lokálna pamäť — pamäťová oblasť prístupná všetkým pracovným jednotkám v rovnakej pracovnej skupine.
- Privátna pamäť — oblasť pamäte prístupná jedinej pracovnej jednotke. Ostatné pracovné jednotky k nej nemajú prístup.

3.4 Programový model

OpenCL podporuje úlohový a dátový paralelizmus, prípadne ich kombináciu, no sústredí sa hlavne na dátový paralelizmus. Úlohový paralelizmus spočíva v rozdelení pôvodnej úlohy na niekoľko menších úloh, ktoré sú ďalej vykonávané paralelne. Každá pracovná jendotka potom vykonáva inú funkciu. Dátovým paralelizmom sa označuje paralelné spracovanie dát na viacerých pracovných jednotkách. Každá pracovná jednotka ma rovnakú funkciu [13].



Obr. 3.2: Pamäťový model [7]

Kapitola 4

Kryptografické hašovacie funkcie

Kryptografická hašovacia funkcia je typ hašovacej funkcie so špeciálnymi vlastnosťami, vďaka ktorým je vhodné ju používať v kryptografii. Hašovacia funkcia je matematický algoritmus, ktorý prevedie dáta ľubovoľnej veľkosti na reťazec dát vopred určenej dĺžky. Tieto funkcie bývajú navrhnuté tak, aby nebolo možné tento reťazec spätne previesť na pôvodné dáta. Správnosť zadaných dát je potom možno overiť len porovnaním hašov. Kryptografická hašovacia funkcia má nasledujúce vlastnosti:

- Je deterministická, čo znamená, že rovnaké dáta vrátia vždy rovnaký haš.
- Pre rôznu veľkosť vstupných dát je dĺžka výstupnej hodnoty funkcie vždy konštantná.
- Nezávisle na veľkosti vstupných dát funkcia je vždy rýchla.
- Pre daný výstup y je výpočetne nemožné nájsť taký vstup x, ktorého výsledný haš je rovný y *(first preimage resistance)*.
- K danému vstupu x nie je výpočetne možné nájsť rozdielny vstup z, ktorého haš by bol zhodný s hašom x *(second preimage resistance)*.
- Je výpočetne nemožné nájsť také dva rozdielne vstupy x a z, ktorých haše by boli zhodné *(collision resistance)*.

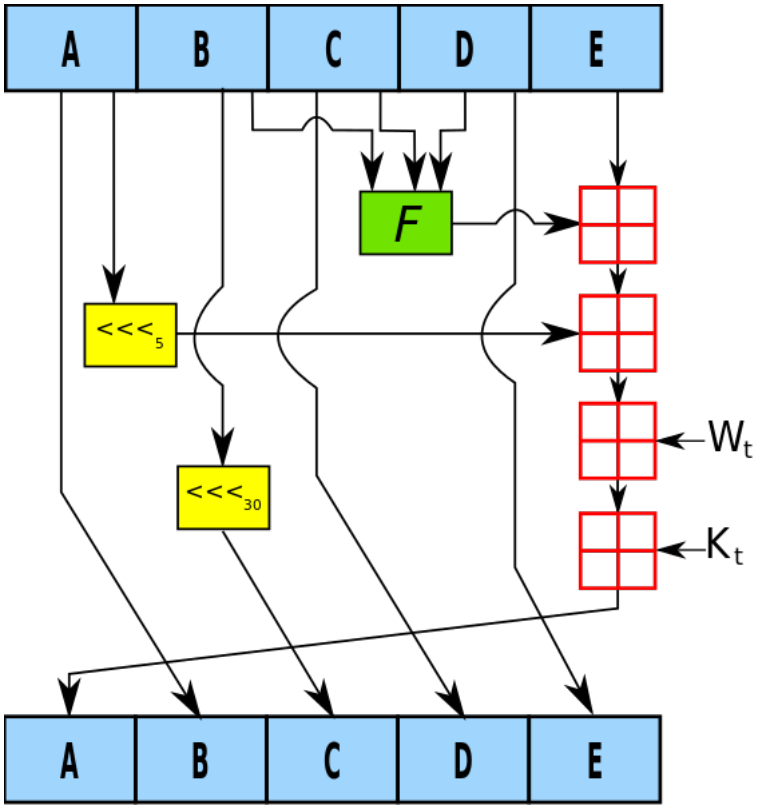
Pre ideálnu hašovaciu funkciu by malo byť možné zistiť pôvodné dáta jedine skúšaním všetkých možných kombinácií (brute-force). Existujú však takzvané *Rainbow Tables*. Sú to tabuľky, ktoré obsahujú predpočítané haše pre rôzne vstupy. Na ochranu proti týmto predpočítaným hodnotám sa k hašovanému vstupu (heslu) pridáva náhodne vygenerovaná bitová hodnota, nazývaná aj soľ. Pridaním tejto hodnoty tabuľka nadobudne rozmery, ktoré ju učinia neefektívnou. Ďalšou vlastnosťou ideálnej hašovacej funkcie je lavínovosť (lavínový efekt). Znamená to, že malá zmena vstupných dát pôvodného hašu spôsobí v konečnom dôsledku tak výraznú zmenu výsledného hašu, že nebude možné nájsť spojitosť medzi týmito hašmi.

4.1 SHA1

Funkcia Secure Hash Algorithm 1 (SHA1) bola navrhnutá a vyvinutá Národnou bezpečnostnou agentúrou (NSA) Spojených štátov amerických. Predchodcom SHA1 je SHA0, publikovaná v roku 1993, od ktorej sa líši v jednej bitovej rotácii. SHA1 vychádza z funkcie

MD5. Prvýkrát bola publikovaná v roku 1995 americkým Národným inštitútom štandardov a technológie (NIST). SHA1 produkuje 160 bitovú hodnotu (haš), v hexadecimálnej sústave, väčšinou s dĺžkou 40 znakov [6].

Na obrázku 4.1 je znázornený priebeh jedného cyklu funkcie. A, B, C, D a E sú 32 bitové registre, obsahujúce vnútorný stav. F je nelineárna meniaca sa funkcia. $\lll_n$ značí bitovú rotáciu vľavo presne o N pozícií. $\boxplus$ je modulo 2^{32}, K_t je konštanta meniaca sa pre každý cyklus t a W_t blok o veľkosti 512 bitov cyklu t, generovaný zo vstupnej správy.



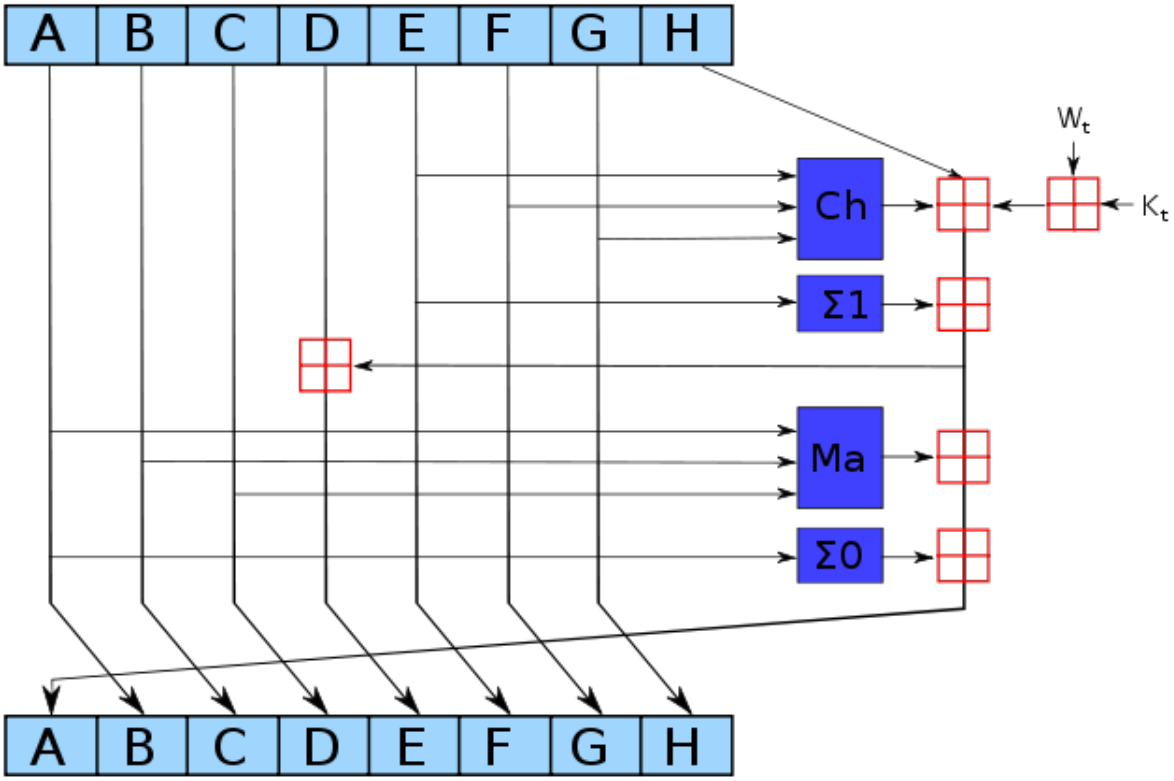
Obr. 4.1: Hašovacia funkcia SHA1[1]

4.2 SHA2

Secure Hash Algorithm 2 (SHA2) je nástupcom SHA1. SHA2 je skupina kryptografických funkcií taktiež navrhnutá NSA a publikovaná v roku 2001. Rozdiel medzi samotnými funkciami je vo veľkosti výsledných hašov. Tieto veľkosti sú 224, 256, 384 a 512 bitov. Do tejto skupiny patria SHA224, SHA256, SHA384, SHA512, SHA512/224 a SHA512/256. Ďalej budú popísané funkcie SHA256 a SHA512, ktoré sú definované pre ODF [6].

Na obrázku 4.2 je znázornený jeden cyklus priebehu funkcií SHA2. A, B, C, D, E, F, G a H sú 32/64 bitové registre obsahujúce vnútorný stav. $\boxplus$ je modulo 2^{32} pre SHA256 a modulo 2^{64} pre SHA512. K_t je konštanta meniaca sa pre každý cyklus t a W_t blok o veľkosti 512/1024 bitov cyklu t, generovaný zo vstupnej správy.

[1]Obrázok prevzatý z — `https://en.wikipedia.org/wiki/SHA-1`



Obr. 4.2: Schéma funkcií SHA2[2]

Ch a **Ma** predstavujú kompresné funkcie pozostávajúce z operácií AND a XOR. Funkcie $\sum 0$ a $\sum 1$ sa skladajú z operácií XOR a bitového posunu vpravo. Rovnica 4.1 bližšie znázorňuje tieto funkcie.

$$\begin{aligned} Ch(E,F,G) &= (E \wedge F) \oplus (\neg E \wedge G) \\ Ma(A,B,C) &= (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C) \\ \textstyle\sum_0(A) &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22) \\ \textstyle\sum_1(E) &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25) \end{aligned} \tag{4.1}$$

4.2.1 SHA256

Maximálna veľkosť vstupnej správy pre SHA256 je $2^{64} - 1$ bitov. Registre A až G, držiace vnútorný stav, sú 32 bitové. Veľkosť výstupu, ako napovedá názov, je 256 bitov. U SHA256 je používané modulo 2^{32}, počet cyklov funkcie je 64.

[2]Obrázok prevzatý z — `https://en.wikipedia.org/wiki/SHA-2`

4.2.2 SHA512

Pre SHA512 je maximálna veľkosť vstupnej správy $2^{128} - 1$ bitov. Osem registrov, držiacich vnútorný stav, je o veľkosti 64 bitov. Veľkosť výstupu je 512 bitov. U SHA512 je používané modulo 2^{64}, počet cyklov funkcie je 80.

4.3 RIPEMD–160

RIPEMD (RACE Integrity Primitives Evaluation Message Digest) je rodina kryptografických hašovacích funkcií, ktoré boli vyvinuté vo výskumnej skupine COSIC v Belgicku. RIPEMD-160 je rozšírená 160 bitová verzia pôvodnej 128 bitovej funkcie RIPEMD. S novou verziou bol zmenený aj počet cyklov funkcie z troch na päť. Algoritmus 1 popisuje priebeh funkcie RIPEMD–160 [3].

Vstup : Dáta
Výstup: Haš vstupných dát
for $i := 0$ *to* $t - 1$ **do**
  $A := h_0; B := h_1; C := h_2; D := h_3; E := h_4;$
  $A' := h_0; B' := h_1; C' := h_2; D' := h_3; E' := h_4;$
  for $j := 0$ *to* 79 **do**
    $T := rol_{s(j)}(A \boxplus f(j, B, C, D) \boxplus X_i[r(j)] \boxplus K(j)) \boxplus E;$
    $A := E; E := D; D := rol_{10}(C); C := B; B := T;$
    $T := rol_{s'(j)}(A' \boxplus f(79 - j, B', C', D') \boxplus X_i[r'(j)] \boxplus K'(j)) \boxplus E';$
    $A' := E'; E' := D'; D' := rol_{10}(C'); C' := B'; B' := T;$
  end
  $T := h_1 \boxplus C \boxplus D'; h_1 := h_2 \boxplus D \boxplus E'; h_2 := h_3 \boxplus E \boxplus A';$
  $h_3 := h_4 \boxplus A \boxplus B'; h_4 := h_0 \boxplus B \boxplus C'; h_0 := T;$
end

Algoritmus 1: Pseudokód hašovacej funkcie RIPEMD-160

h_0 až h_4 na začiatku cyklu obsahujú inicializačné hodnoty. Ich hodnota sa mení každou iteráciou. t značí počet cyklov, čiže päť. $X_i[j]$ sú vstupné dáta v piatich 32 bitových blokoch, rozdelených po dvoch bajtoch. Zároveň platí: $0 \leq i \leq t - 1$ a $0 \leq j \leq 15$. s a s' označujú počet pozícií o ktoré sa rotuje. Obe sú premenlivé podľa danej iterácie. $\boxplus$ je modulo 2^{32}, rol_s je bitová rotácia vľavo o s pozíciách. A, B, C, D, E, a T sú pomocné premenné.

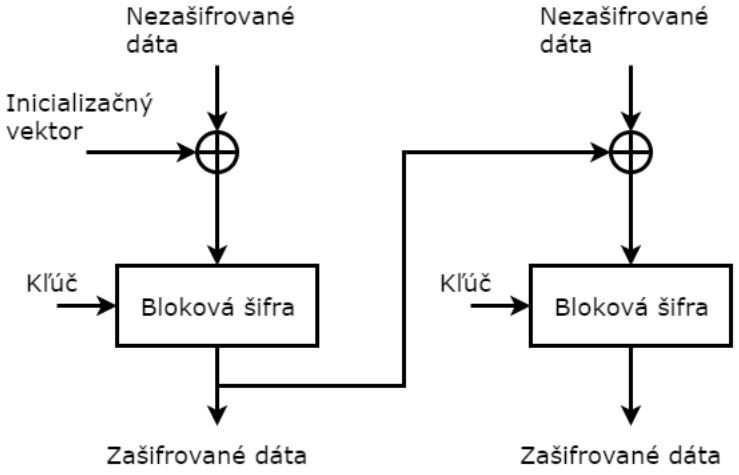
Kapitola 5

Šifrovacie algoritmy

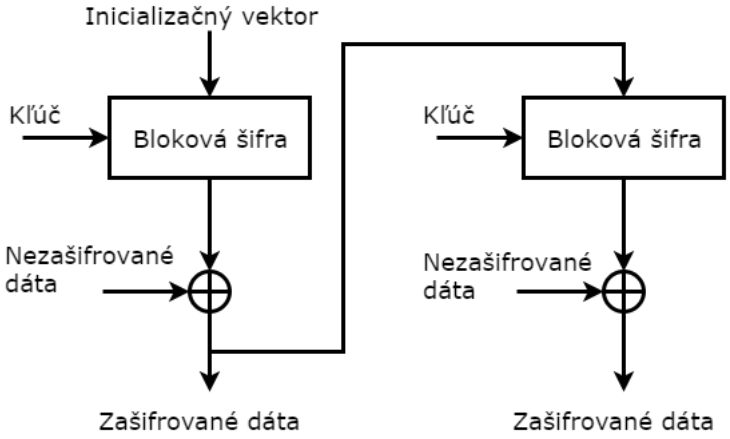
Kryptografia (z gréckeho kryptós a gráphein, t.j. skrytý a písať) študuje metódy utajovania zmyslu správ ich prevodom do formy, ktorú nie je možné rozlúštiť, pokiaľ nie je známy spôsob tohto prevodu. Spočiatku šifrovanie prebiehalo veľmi jednoducho pomocou ceruzky a papiera, prípadne s využitím špeciálnych nástrojov, vytvorených pre dané šifry.

V tejto kapitole bude venovaná pozornosť blokovým šifrám. Blokové šifry spracúvajú dáta v častiach (blokoch) určitej veľkosti. Preto musia byť tieto dáta najprv rozdelené do rozdielnych blokov. Následný proces zašifrovania sa delí podľa postupu zaobchádzania s blokmi. Najjednoduchším príkladom je takzvaný režim *electronic codebook* (ECB), ktorý každý blok zašifruje nezávisle na ostatných. Kvôli nedostatkom tohto režimu boli navrhnuté nové režimy ako napríklad *cipher block chaining* (CBC) alebo *cipher feedback* (CFB). V oboch prípadoch je použitý inicializačný vektor (IV), aby sa zabránilo rovnakým výstupom pri rovnakých vstupoch.

V režime CBC je IV skombinovaný s nezašifrovanými dátami prvého bloku. Po zašifrovaní je tento blok skombinovaný operáciou XOR s následujúcim blokom rovnako ako IV v prvom bloku. Režim CFB najprv zašifruje IV, ktorý skombinuje s nezašifrovaným blokom pomocou operácie XOR. Výsledok je potom dalej použitý ako IV pre ostatné bloky. Na obrázkoch 5.1 a 5.2 je možné vidieť priebeh oboch režimov.



Obr. 5.1: Režim cipher block chaining (CBC) blokových šifier



Obr. 5.2: Režim cipher feedback (CFB) blokových šifier

5.1 Blowfish

Blowfish je symetrická bloková šifra, ktorú navrhol v roku 1993 Bruce Schneier ako alternatívu k algoritmu DES. Počet iterácií šifrovacej funkcie je 16, používa bloky o veľkosti 64 bitov a kľúč môže dosahovať dĺžky 32 až 448 bitov. Napriek zložitejšej inicializácii pred samotným šifrovaním, dosahuje stále veľmi efektívne výsledky. Blowfish používa Feistelovu šifru [9, 12].

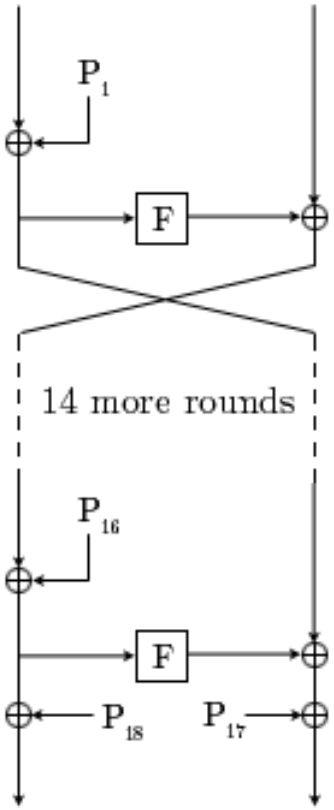
Na obrázku 5.4 je znázornený samotný priebeh algoritmu Blowfish. V 256 hodnotových S-boxoch a 18 hodnotových P–poliach uchováva algoritmus dve polia podkľúčov. V každej iterácii je použitá jedna hodnota z P–poľa a po poslednej iterácii sú obidve polovicw dát bloku zlúčené funkciou XOR s jednou z nepoužitých hodnôt. Na obrázku 5.5 je zobrazená funkcia F, ktorá rozdelí 32 bitový vstup na štyri 8 bitové, ktoré sú nakoniec po úprave modulom 2^{32} naspäť spojené funkciou XOR.

5.2 DES

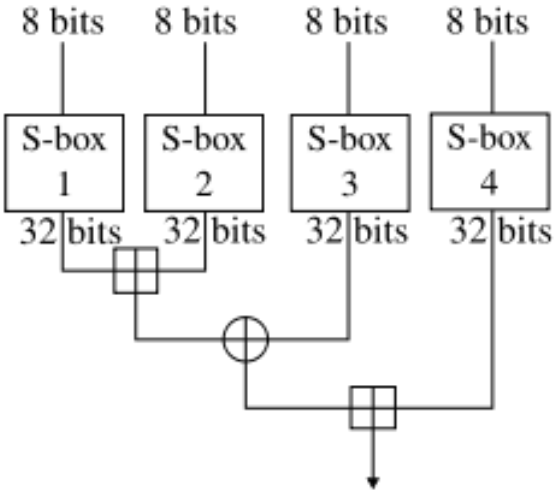
Data Encryption Standard (DES) je symetrická bloková šifra, založená na jej predchodcovi – šifre Lucifer [9]. Vyvinutá bola firmou IBM a na jej vývoji sa podieľal aj pôvodný autor šifry Lucifer – Horst Feistel. Prvýkrát bola publikovaná v roku 1975 a prijatá ako štandard pre americkú vládu v roku 1977.

DES používa bloky o veľkosti 64 bitov a celkom 16 iterácií algoritmu. Kľúč používaný na transformáciu vstupného reťazca je veľký 64 bitov, no z toho len 56 je využitých v priebehu algoritmu. Ďalších 8 slúži len na kontrolu parity, čiže efektívna dĺžka kľúča je 56 bitov. Pred prvou iteráciou je blok rozdelený na dve 32 bitové časti, ktoré sú ďalej striedavo spracovávané. Vďaka tomuto spôsobu striedania, sa šifrovanie a dešifrovanie týmto algoritmom, líši len v poradí podkľúčov.

Feistelova funkcia, používaná v algoritme DES, je znázornená na obrázku 5.3. Funkcia pracuje s jednou polovicou bloku a podklúčom. 32 bitová polovica bloku je najprv rozšírená na 48 bitov (Expansion), potom pomocou operácie XOR ($\oplus$) je spojená s podkľúčom. V ďalšom kroku je tento blok rozdelený na osem 6 bitových častí, z ktorých každá prejde substitúciou (S1-S8), kde je 6 vstupných bitov zamenených za 4 výstupné podľa tabuľky. Posledným krokom je zmena poradia bitov podľa vopred určenej permutácie. [4]

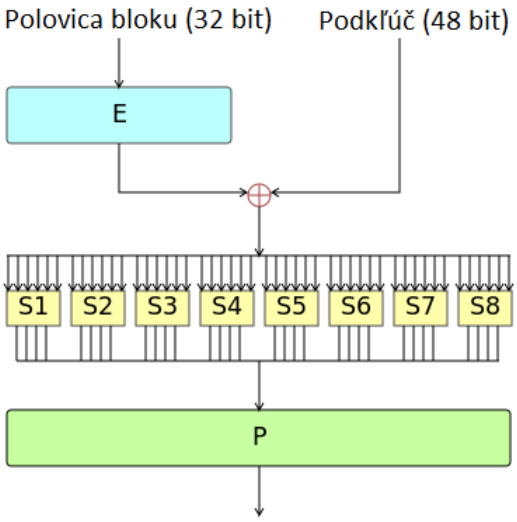


Obr. 5.4: Algoritmus Blowfish[2]



Obr. 5.5: Funkcia F (Feistelova) algoritmu Blowfish[2]

[2]Obrázok prevzatý z – `https://cs.wikipedia.org/wiki/Blowfish`



Obr. 5.3: Feistelova funkcia použitá algoritmom DES[1]

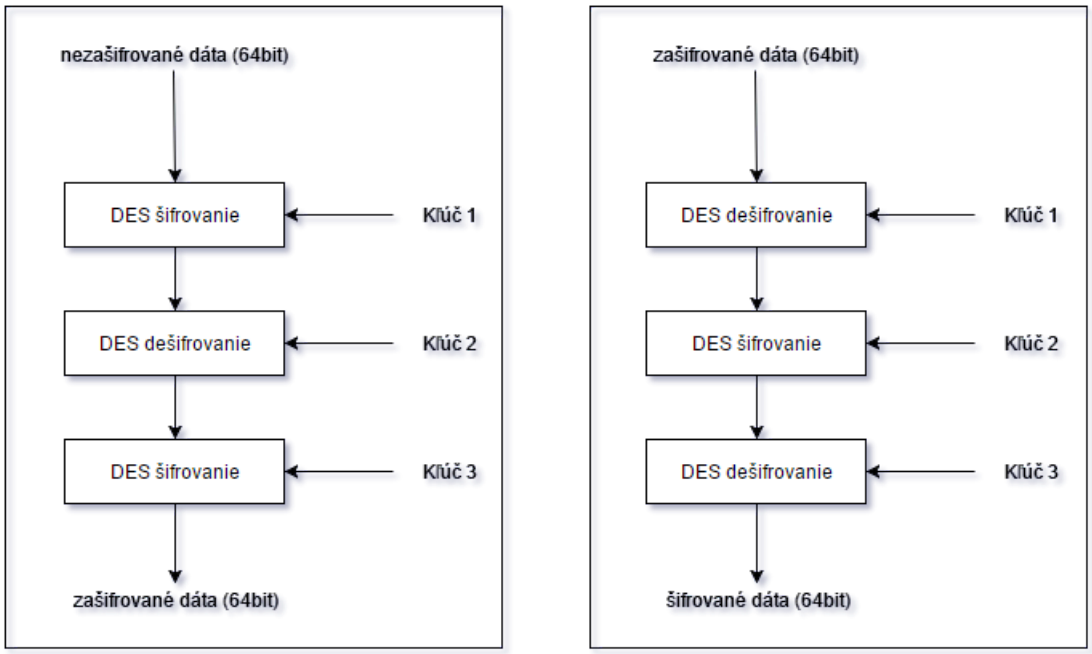
5.2.1 Triple DES

Triple DES (3DES), alebo Triple Data Encryption Algorithm (TDEA), je rovnako ako DES symetrická bloková šifra, ktorá spočíva v aplikovaní algoritmu DES trikrát. Po zistení nedostatkov algoritmu DES bol 3DES navrhnutý ako náhrada. Použitím pôvodného algoritmu viackrát, sa dosiahla väčšia dĺžka kľúča, bez potreby navrhnutia celkom nového algoritmu. 3DES je však oveľa pomalší oproti novým algoritmom, ako napríklad AES, ktorý je popísaný v sekcii 5.3 [1].

Na obrázku 5.6 je zobrazený postup algoritmu 3DES pri šifrovaní a aj spätnom dešifrovaní dát. Pri každom použití algoritmu DES je použitý kľúč (K_i) a na základe jeho vzťahu k ostatným kľúčom (K_1, K_2, K_3) ide o jeden z troch variantov:

- **Všetky kľúče sú na sebe nezávislé** -– najsilnejší variant, dĺžka kľúča je 168 bitov.
- **Kľúče K1 a K2 sú nezávislé,** $K3 = K1$ — slabší variant, no stále silnejší než samotný DES, dĺžka kľúča je 112 bitov.
- **Všetky kľúče sú identické, K1 =** $K2 = K3$ — štandardný DES, umožňuje spätnú kompatibilitu, dĺžka kľúča je 56 bitov.

[1]Obrázok prevzatý z – `https://en.wikipedia.org/wiki/Data_Encryption_Standard`



Obr. 5.6: Priebeh algoritmu 3DES pri šifrovaní (vľavo) a dešifrovaní (vpravo)

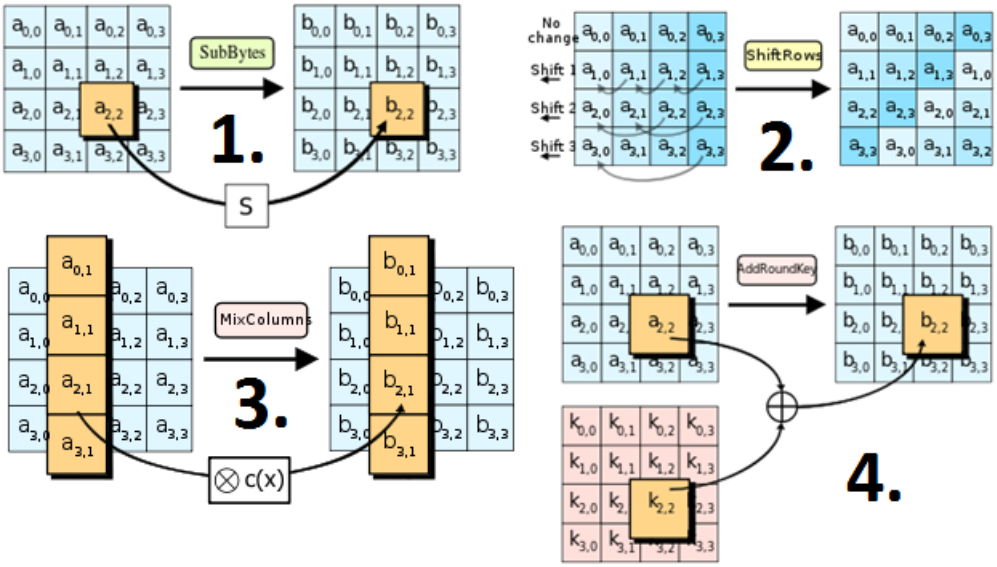
5.3 AES

Algoritmus DES a jeho dĺžka kľúča bola nedostačujúca a preto bolo potrebné nájsť kvalitnejšiu náhradu. Jedným z možných riešení bol 3DES, spomenutý v sekcii 5.2.1, ktorý však nebol dosť rýchly a preto bol navrhnutý Advanced Encryption Standard (AES). Americká NIST spomedzi pätnástich navrhnutých šifier vybrala pre AES šifru Rijndael, ktorú vytvorili Joan Daemen a Vincent Rijmen [5].

Dĺžka bloku je stála a to 128 bitov. Veľkosť kľúča môže byť 128, 192 alebo 256 bitov (AES 128, AES 192, AES 256). Počet iterácií je taktiež stály a závisí od dĺžky kľúča,10 iterácií pre kľúč dĺžky 128 bitov, 12 pre 192 bitový kľúč a 14 iterácií s 256 bitovým kľúčom. AES pracuje nad stĺpcovo orientovanou maticou o rozmeroch $4 * 4$. Algoritmus je možné rozdeliť do štyroch častí, ktoré sú zobrazené aj na obrázku 5.7:

1. Odvodenie podkľúčov zo šifrovacieho kľúča pomocou vlastnej funkcie[3].
2. Každý bajt matice je skombinovaný pomocou operácie XOR s podkľúčom (AddRoundKey).
3. Pre každý cyklus — nelineárne nahradenie každého bajtu podľa tabuľky (SubBytes), posunutie hodnôt na riadku o určitý počet pozícií (ShiftRows), hodnoty v stĺpcoch matice sú násobené fixnými hodnotami (MixColumns). Nakoniec sa spraví znovu 2. bod (AddRoundKey).
4. V poslednej iterácii sa hodnoty už nenásobia.

[3]Zdroj: `http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf`



Obr. 5.7: Kroky algoritmu AES. SubBytes (1.), ShiftRows (2.), MixColumns (3.), AddRoundKey (4.)[5]

[5]Obrázok prevzatý z – `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`

Kapitola 6

OpenDocument

OpenDocument, alebo Open Document Format (ODF), celým názvom OASIS[1] Open Document Format for Office Applications, je súborový formát, určený pre textové dokumenty, tabuľky, grafy, prezentácie a databázy. Formát OpenDocument bol vytvorený technickou komisiou OASIS, za účelom sprístupnenia nového otvoreného formátu, založeného na značkovacom jazyku XML, verejnosti. Samotný formát vznikol zo súborového formátu OpenOffice.org XML, vyvinutého firmou Sun Microsystems ako štandard pre dokumenty. V tabuľke 6.1 sa nachádza zoznam typov súborov formátu ODF a ich príslušných súborových prípon [2].

Obsahu súboru	Prípona	Prípona (šablóna)
text	.odt	.ott
tabuľkový procesor	.ods	.ots
prezentácia	.odp	.otp
kresba	.odg	.otg
graf	.odc	.otc
vzorec	.odf	.otf
obrázok	.odi	.oti
Master Document	.odm	—
databáza	.odb	—
šablóna webovej stránky	—	.oth

Tabuľka 6.1: Prípony súborov formátu ODF

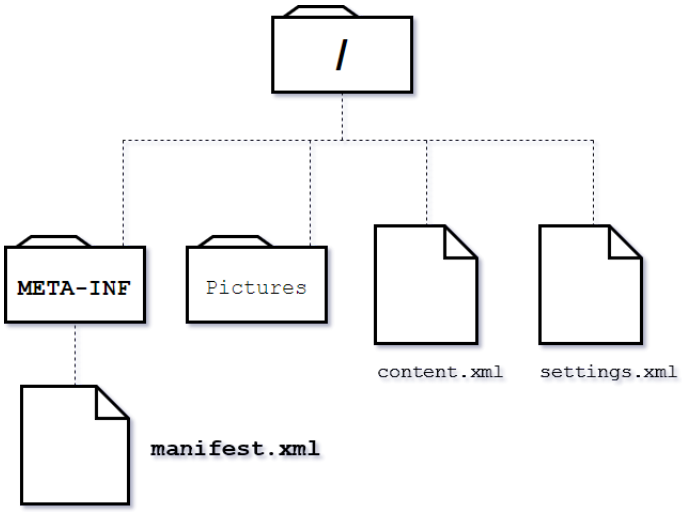
6.1 Štruktúra ODF

Súbor vo formáte OpenDocument sa na prvý pohľad javí ako jeden samostatný súbor, no nemusí tomu tak byť. Jednou z foriem ODF súboru je jediný XML súbor, nazývaný aj ako Flat XML. Tento spôsob ukladá všetky dáta do jedného súboru. Druhou možnosťou je rozdelenie všetkých dát do viacerých súborov (tie môžu byť ďalej umiestnené v priečinkoch) podľa ich významu. Samotný obsah textového dokumentu je uložený samostatne, zatiaľ čo väčšina informácií o vzhľade, formátovaní a použitých štýloch sa nachádza v ďalšom súbore.

[1] Organization for the Advancement of Structured Information Standards (OASIS) - `https://www.oasis-open.org/`

Tieto súbory sú potom zabalené do ZIP archívu (prípadne JAR). Názornú ukážku takto štruktúrovaného súboru je možné vidieť na obrázku 6.1.

Medzi základné súbory patria content.xml, styles.xml, settings.xml, meta.xml, mimetype a manifest.xml. Popisom obsahu zmienených súborov sa zaoberá nasledujúca sekcia 6.1.1.



Obr. 6.1: Štruktúra ODF súboru (ZIP)

6.1.1 Obsah súborov

Keďže sa jedná o XML súbory, každý súbor bude pozostávať z jednotlivých elementov. Pokiaľ sa nejedná o balík súborov v ZIP (JAR) archíve, ale o jednotlivý XML súbor, bude tento súbor mať koreňový element *<office:document>*. V opačnom prípade budú mať súbory rozdielne koreňové elementy, ktoré budú jednoznačne určovať ich nasledujúci obsah.

- **content.xml** – koreňovým elementom je *<office:document-content>*, súbor obsahuje samotné dáta (napr. text pri textovom dokumente) a automatické štýly, ktoré boli použité.
- **styles.xml** – OpenDocument používa na formátovanie štýly a obsahom práve tohto súboru je väčšina použitých štýlov. Obsahuje tiež informácie o automatických štýloch použitých pri tvorbe týchto štýlov. Koreňový element je *<office:document-styles>*.
- **meta.xml** – súbor s koreňovým elementom *<office:document-meta>* obsahujúci meta dáta ako napr. dátum vytvorenia a poslednej úpravy alebo meno autora.
- **settings.xml** – obsahuje nastavenia špecifické pre aplikáciu, v ktorej bol dokument vytvorený a koreňovým elementom tohto súboru je *<office:document-settings>*.

S ohľadom na cieľ tejto práce je najdôležitejší súbor **manifest.xml**. Tento súbor sa nenachádza v koreňovom adresári ako predchádzajúce súbory, ale vždy v priečinku META-INF. Obsahom súboru sú informácie o všetkých súboroch OpenDocument balíčku a pokiaľ je dokument zašifrovaný, obsahuje taktiež informácie potrebné na jeho dešifrovanie. Podrobnejší popis súboru manifest.xml sa nachádza v sekcii 6.1.2.

6.1.2 Manifest.xml

V prípade, že je dokument zašifrovaný, sú v tomto súbore uložené všetky potrebné informácie na spätnú dešifráciu. V tejto sekcii budú postupne rozpísané jednotlivé elementy, ktoré súbor *manifest.xml* môže obsahovať. Samotné šifrovanie je zahrnuté v sekcii 6.2.

- **<manifest:manifest>** je koreňovým elementom každého *manifest.xml* súboru a jeho atribút *manifest:version* určuje verziu daného dokumentu.
- **<manifest:file-entry>** špecifikuje jeden súbor z balíku dokumentu. Pokiaľ je súbor šifrovaný, obsahuje atribút *manifest:size*, ktorého hodnotou je veľkosť nešifrovaného, nekomprimovaného súboru a je prítomný aj jeho dcérsky element **<manifest:encryption-data>**.
- **<manifest:encryption-data>** obsahuje spolu so svojimi atribútmi a dcérskymi elementmi informácie o procese a parametroch šifrovania súboru. Atribút *manifest:checksum-type* ukazuje aký typ hašovacej funkcie bol použitý a hodnota *manifest:checksum* je výsledný kontrolný súčet, ktorý nám umožní overiť správnosť hesla.
- **<manifest:algorithm>** a presnejšie jeho atribút *manifest:algorithm-name* určujú meno algoritmu, ktorým boli zašifrované dáta. Hodnota atribútu *manifest:initialisation-vector* je hodnotou inicializačného vektoru, vo forme reťazca v kódovaní BASE64[2], ktorý použije daný algoritmus.
- **<manifest:start-key-generation>** špecifikuje ako vznikol počiatočný kľúč zo zadaného hesla. *manifest:start-key-generation* určuje algoritmus, ktorý bude použitý na generovanie počiatočného kľúča. Atribút *manifest:key-size* definuje dĺžku kľúča, ktorý vráti algoritmus špecifikovaný predchádzajúcim atribútom. Tento element nie je povinný a pokiaľ prítomný v súbore nie je, použijú sa implicitné hodnoty – SHA1 pre algoritmus a 20 pre dĺžku kľúča.
- **<manifest:key-derivation>** určuje ako bol derivačný kľúč vypočítaný z počiatočného kľúča. *manifest:iteration-count* definuje počet iterácií pre algoritmus. Atribútom *manifest:key-derivation-name* je zadaný algoritmus použitý na deriváciu kľúča. *manifest:salt* nesie binárnu hodnotu v kódovaní base64, tzv. kryptografickú soľ, ktorá nemá maximálnu dĺžku. Atribút *manifest:key-size* je dĺžkou kľúča vráteného algoritmom na deriváciu kľúča. Implicitná hodnota pre tento atribút je 16.

6.2 Šifrovanie ODF

Pri zašifrovanom ODF súbore nemusia byť všetky súbory zašifrované, napr. manifest.xml nesmie byť nikdy zašifrovaný, pretože ako už bolo spomenuté v predchádzajúcej sekcii 6.1.2, obsahuje informácie o samotnom šifrovaní súborov. Každý súbor, ktorý bude šifrovaný je najprv komprimovaný algoritmom DEFLATE[3] (bezstratová kompresia) a má nastavený príznak STORED namiesto DEFLATED v centrálnom adresári súboru ZIP. Veľkosť pôvodného nezašifrovaného súboru, ktorá je uložená v záznamoch centrálneho adresára, by mala byť prepísaná novou veľkosťou komprimovaného a zašifrovaného súboru.

[2] `https://www.ietf.org/rfc/rfc4648.txt`
[3] `https://www.ietf.org/rfc/rfc1951.txt`

Samotný proces šifrovania OpenDocument súboru je možné popísať v troch fázach. Na šifrovanie možno použiť viacero algoritmov a hašovacích funkcií, ktoré sú podrobnejšie rozobrané v kapitolách 4 a 5 V nasledujúcom popise týchto fáz bude použitá hašovacia funkcia SHA1 a algoritmy PBKDF2 a Blowfish.

V prvej fáze je vygenerovaný počiatočný kľúč zo zadaného hesla v kódovaní UTF-8[4]. Tento kľúč je 20 bajtový kontrolný súčet z funkcie SHA1, ktorej bolo predložené zadané heslo. Ďalej nasleduje derivácia tohto kľúča pre každý súbor, ktorý bude šifrovaný. Algoritmus PBKDF2, využívajúci funkciu HMAC-SHA-1, je použitý na spomenutú deriváciu kľúča. Pre každý jednotlivý súbor je vygenerovaná náhodná 16 bajtová hodnota (soľ), ktorá po spojení s počiatočným kľúčom a ich následnej derivácii, tvorí unikátny 128 bitový kľúč pre súbor. Štandardný počet iterácií pre tento proces je 1024. Na záver sú súbory zašifrované. Inicializačný vektor, t.j. náhodne vygenerovaná 8 bajtová hodnota, je spolu s derivovaným kľúčom použitý na zašifrovanie obsahu súboru algoritmom Blowfish v 8 bitovom móde CFB.

Obdobne je možné namiesto Blowfish použiť algoritmus AES alebo 3DES v režime CBC, ktoré sú uvedené v špecifikácii ODF ako podporované algoritmy. 3DES napriek tomu nevyužíva žiadny známy editor. Funkciu SHA1 je tiež možné nahradiť funkciami SHA256, SHA512 alebo RIPEMD-160. Ani RIPEMD-160, rovnako ako 3DES, nie je známymi editormi využívaný.

[4] `https://tools.ietf.org/html/rfc3629`

Kapitola 7

Návrh modulu

Cieľom tejto práce je rozšíriť nástroj Fitcrack o ďalší modul, ktorý umožní obnovovanie hesiel z formátu OpenDocument. V predchádzajúcich kapitolách bol popísaný samotný nástroj Fitcrack, štandard OpenCL, hašovacie funkcie a šifrovacie algoritmy, ktoré môžu byť použité a samotná štruktúra súboru formátu OpenDocument. Na základe informácií o nástroji Fitcrack a súboroch formátu OpenDocument bude v tejto kapitole popísaný návrh modulu ODF a postup ako získava potrebné informácie o šifrovaní.

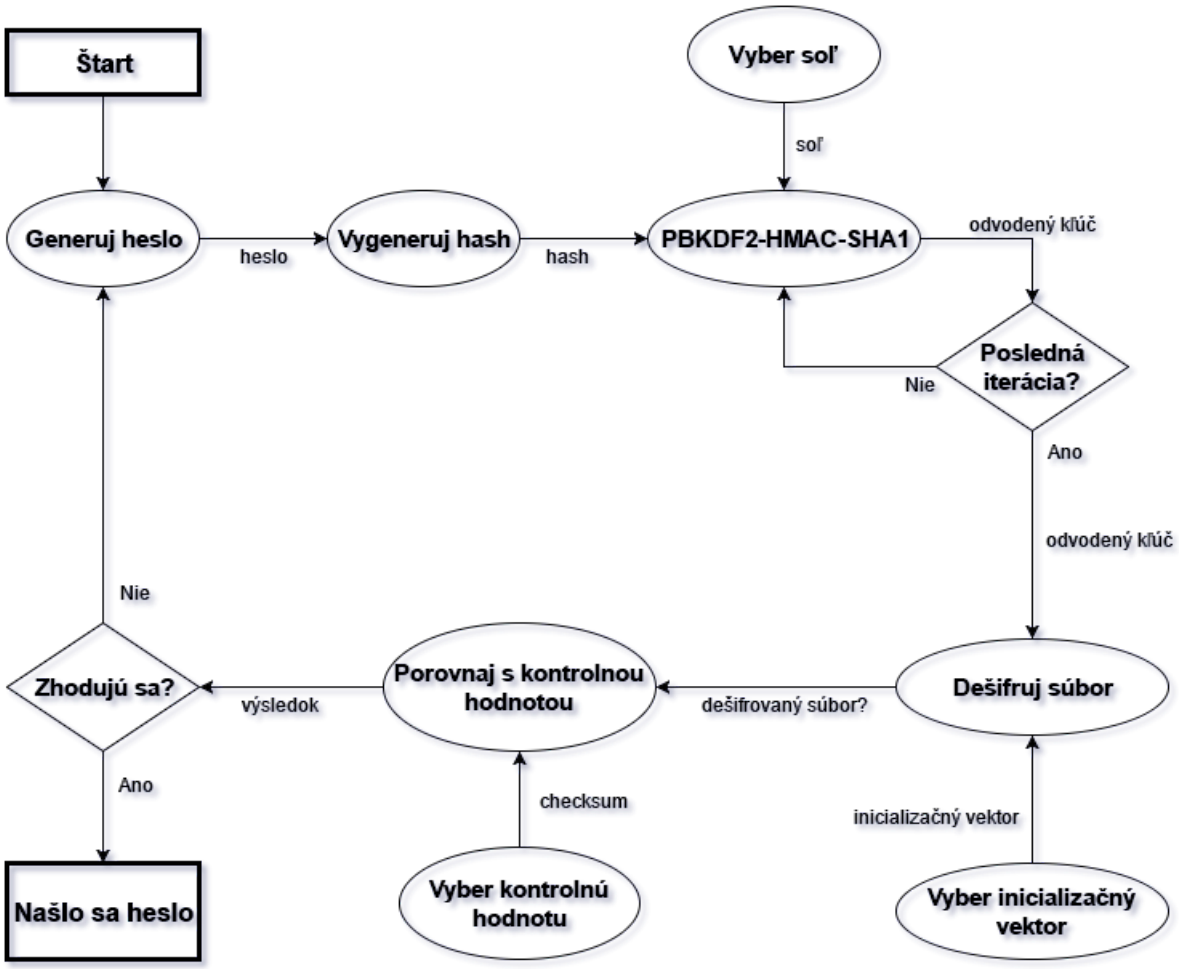
Keďže súbory ODF sú vo svojej podstate väčšinou archívmi ZIP formátu, pri overovaní správnosti formátu je nutné porovnávať signatúru súboru so signatúrou **50 4B 03 04**, čo je jedna z možných signatúr. Taktiež je dôležité zohľadniť možnosť, že zadaný súbor nemusí byť formátu ODF. Kontrola prípony zadaného súboru spolu s kontrolou signatúry je teda najbezpečnejšou metódou overenia formátu. Predíde sa tak situácii, kedy by sa modul pokúšal získať informácie z archívu ZIP, ktorý nie je súborom formátu ODF. Tabuľka 6.1 obsahuje všetky definované prípony pre ODF súbor.

Počas štúdia špecifikácií ODF a dostupných editorov, ktoré tento formát podporujú, bolo zistené že nie všetky dovoľujú program aj zašifrovať. Aplikácie ako napríklad AbiWord, Google Docs, Microsoft Word podporujú čítanie súborov ODF, no iba nezašifrovaných a neumožňujú pri ukladaní do formátov ODF súbor zašifrovať. Jediné programy, ktoré podporovali aj šifrovanie, boli LibreOffice a Apache OpenOffice. LibreOffice pri tom používa hašovaciu funkciu SHA256 spolu s algoritmom AES v režime CBC. Apache OpenOffice oproti tomu používa kombináciu SHA1 a Blowfish v osem bitovom režime CFB. Z tohto dôvodu boli teda vynechané funkcie a algoritmy *SHA512*, *RIPEMD-160* a *3DES* napriek tomu, že implementácia *SHA512* je už prítomná v nástroji Fitcrack. Ďalej, ako už bolo spomenuté v sekcii 6.1, sa potrebné informácie na obnovu hesla nachádzajú v súbore vo vnútri tohto archívu. Archív je nutné rozbaliť a zo súboru vyextrahovať tieto informácie.

Kontrolný súčet potrebný na overenie správnosti hesla a typ hašovacej funkcie nájdeme v atribútoch *manifest:checksum* a *manifest:checksum-type*. Inicializačný vektor sa nachádza v atribúte *manifest:initialisation-vector*, zatiaľ čo informácia o použitom šifrovacom algoritme je zaznamenaná v atribúte *manifest:algorithm-name*. Algoritmus generujúci počiatočný kľúč je hodnotou atribútu *manifest:start-key-generation*, zatiaľ čo dĺžka kľúča, ktorý vráti, je v atribúte *manifest:key-size*. Informáciu o derivácií kľúča extrahujeme z atribútov *manifest:key-derivation-name*, *manifest:salt*, *manifest:key-size*. Tieto atribúty obsahujú meno algoritmu použitého na deriváciu kľúča, kryptografickú soľ a veľkosť kľúča vráteného daným algoritmom. Z hodnoty atribútu *manifest:iteration-count* získame počet iterácii.

V momente, kedy sú všetky informácie vyextrahované zo súboru, úloha FitXtractoru končí a vygeneruje sa súbor v XML formáte pre Fitcrack. Nasleduje spustenie generá-

tora hesiel a pokus o obnovu hesla. Na zistenie správnosti je potrebné, aby vygenerované heslo prešlo rovnakým postupom šifrovania ako heslo pôvodné. Po odvodení kľúča sa modul pokúsi dešifrovať súbor a vypočítať z neho kontrolnú hodnotu. Nasleduje porovnanie s kontrolnou hodnotou získanou zo súboru manifest. Pokiaľ sa hodnoty zhodujú, vygenerované heslo je správne. V opačnom prípade sa začína odznova s ďalším vygenerovaným heslom. Celý proces je znázornený na obrázku 7.1.



Obr. 7.1: Proces obnovy hesla formátu OpenDocument

Kapitola 8

Implementácia

Táto kapitola zahŕňa implementáciu modulu, ktorého návrh bol popísaný v kapitole 7. Kapitola je rozdelená do dvoch častí, kde prvá zahŕňa implementáciu časti modulu pre extrakciu všetkých potrebných informácií zo súboru, zatiaľ čo druhá časť obsahuje dešifrovaciu časť tohoto modulu. V kapitole budú taktiež, na príslušných miestach pre časti modulu, spomenuté použité knižnice.

8.1 Extrakcia

Pri implementácii bolo potrebné rozšíriť funkčnosť nástroja FitXtractor o možnosť extrakcie dát zo súborov formátu OpenDocument (ODF). Za týmto účelom bola vytvorená trieda *Odf* rozširujúca triedu *XtractorFile*. Ako už bolo spomenuté v sekcii 6.1.2, informácie o použitom šifrovaní sú uložené v súbore formátu XML, preto je potrebné tieto informácie spracovať do formy vhodnej pre nasledujúcu dešifráciu. Pred samotným spracovaním súboru je potrebné ho extrahovať z balíku ZIP (JAR). K tomu je použitá metóda *prepareOutput*, ktorá najprv skontroluje, či má súbor správny formát. Pokiaľ signatúra súboru zodpovedá signatúre ZIP arhívu, skontroluje, či prípona súboru je zhodná s jednou z definovaných prípon pre súbory ODF (tabuľka 6.1). Po úspešnej kontrole je za pomoci knižnice *libzip* vybraný súbor *manifest* a jeden ďalší súbor, ktorého obsah bude použitý pri dešifrácii a následnej kontrole správnosti hesla. Týmto súborom by mohol byť napríklad súbor *content.xml*, ktorý vždy obsahuje dáta nezávisle na type súboru. Jeho veľkosť je však premenlivá a môže nadobúdať hodnoty, ktoré by mohli proces značne spomaliť. Napríklad pri variante použitia hašovacej funkcie na celý obsah súboru, s cieľom vytvorenia kontrolnej hodnoty. Z tohto dôvodu sa využívajú súbory *meta.xml*, alebo *manifest.rdf*.

Na spracovanie dôležitých informácií boli vytvorené metódy *prepareOutput*, *getElement*, *parseElement*, *formatElement*. Metóda *prepareOutput* zastrešuje celý proces úpravy textovej podoby obsahu súboru. Cyklickým volaním metódy *getElement* sa spracúvajú atribúty elementov na žiadosť metódy *prepareOutput*. K spracovaniu atribútov sú využívané posledné dve metódy – *parseElement* a *formatElement*. Prvá odstráni prebytočné znaky z reťazca a vráti len podstatnú hodnotu, zatiaľ čo druhá metóda, na základe týchto informácií, uloží a nastaví príslušné hodnoty. Po dokončení extrakcie všetkých elementov a ich následnom vložení do štruktúry je obsah vybraného súboru, ktorý bude dešifrovaný, zakódovaný funkciou *encode64* do formátu base64. Táto funkcia je prebratá zo skorších implementácií modulov pre Fitcrack.

8.2 Dešifrovanie

Rovnako ako pre nástroj FitXtractor, tak aj pre Fitcrack bola vytvorená nová trieda *Odf*, ktorá rozširuje triedu *Cracker*. Jej funkciou je dešifrovanie dát, ktoré dostane spolu s informáciami o použitom šifrovaní vo forme XML súboru. Ten je výstupom nástroja FitXtractor z predchádzajúceho kroku. O načítanie údajov z tohto súboru do asociatívneho poľa s položkami, pomenovanými podľa mien elementov, sa postará nástroj Fitcrack. Konštruktor triedy potom inicializuje vlastnú štruktúru s dátami príslušnými hodnotami z tohto poľa.

8.2.1 CPU

Verziu CPU dešifrácie ODF súborov obsahuje metóda *checkPassword*. Postupuje tak ako bolo popísané v návrhu modulu v predchádzajúcej kapitole 7. V prvom kroku sa zadané heslo, vygenerované príslušným generátorom, predá hašovacej funkcii. Podporované funkcie sú *SHA1*, *SHA256* a *SHA512*, ktoré boli implementované predchádzajúcimi modulmi. *RIPEMD-160* podporovaný nie je. Výsledný haš je pomocou funkcie *PBKDF2-HMAC-SHA1* rozšírený o soľ, po danom počte iterácií. Po tomto kroku nasleduje dešifrovanie obsahu súboru a to buď jedným z variantov algoritmu *AES* (128, 192, 256) v CBC režime alebo algoritmom *Blowfish* v 8 bitovom CFB režime. Implementácia algoritmu *AES* je prítomná z predošlých modulov a pre algoritmus *Blowfish* bola prevzatá verejne dostupná implementácia[1]pod licenciou Ms-PL (Microsoft Public License). Podpora pre *3DES* nebola implementovaná, ako už bolo spomenuté v sekcii 6.2. Keďže neboli nájdené programy, ktoré by tento algoritmus podporovali, nemohla byť jeho funkčnosť otestovaná. To isté platí pre funkciu RIPEMD-160.

Po dokončení dešifrovania dát je využitím hašovacej funkcie z výsledku vypočítaná kontrolná hodnota. Opäť môže byť použitý jeden z variantov hašovacích funkcií (*SHA1*, *SHA256*, *SHA512*), no na rozdiel od použitia pre vytváranie hašu z hesla, je vstup mnohonásobne väčší. Z tohto dôvodu môže byť kontrolná suma počítaná len z prvých 1024 bajtov súboru.

8.2.2 GPU

Metódou *assignGPU* je na CPU zahájená príprava crackovania s pomocou GPU. V prvom rade sa skontroluje, rovnako ako pri CPU verzii, či sú použité šifrovacie algoritmy a hašovacie funkcie podporované. V ďalšom kroku sa nastaví meno súboru a kernelu na základe použitého algoritmu. Program pokračuje alokovaním pamäte pre potrebné parametre kernelu:

- **data buffer** — Obsahuje kompletný obsah zašifrovanej časti súboru alebo len prvých 1024 bajtov, v závislosti na použití hašovacej funkcie.
- **global buffer** — Predstavuje globálnu pamäť pre dešifrované dáta.
- **content length buffer** — Tento parameter nesie informáciu o pôvodnej veľkosti zašifrovaných dát. Túto veľkosť môže byť potrebné na GPU upraviť pri použití algoritmu *Blowfish* na násobok čísla 8, pretože implementácia algoritmu pracuje v 8 bitovom režime a kontroluje veľkosť bloku.

[1]Zdroj: `https://www.codeproject.com/Articles/1400/A-C-Implementation-of-the-Blowfish-Encryption-Decr`

- **iter count buffer** — Parameter s počtom iterácií pre *PBKDF2*.
- **iv buffer** — Nesie inicializačný vektor, potrebný pre šifrovací algoritmus.
- **salt buffer** — Obsahuje kryptografickú soľ, ktorá bude použitá funkciou *PBKDF2* na rozšírenie hesla.
- **checksum buffer** — Týmto parametrom je predávaná kontrolná hodnota, s ktorou sa bude porovnávať hash z dešifrovaných dát.
- **algorithm buffer** — Označuje, ktorá verzia algoritmu sa použie. Použitý je iba pri algoritme *AES*.
- **left** a **right** — Pre algoritmus *Blowfish* je inicializačný vektor potrebné najprv upraviť do podoby hexadecimálneho čísla, rozdeliť na dve polovice a poslať túto číselnú hodnotu kernelu. Parameter **left** obsahuje ľavú a parameter **right** pravú polovicu inicializačného vektoru. Tieto parametre sú použité iba pri algoritme *Blowfish*.

Po alokovaní potrebnej pamäte sú jednotlivé parametre kernelu inicializované príslušnými hodnotami. Nasledujúcim krokom je inicializácia kernelu a nastavenie jeho parametrov.

Proces dešifrácie súboru a zistenia hesla na GPU sa oproti CPU verzii líši minimálne. Jediným výrazným rozdielom je, že GPU verzia podporuje iba dve varianty, ktoré sú obsiahnuté v kerneloch *odf_aes_sha256_kernel* a *odf_blowfish_kernel*. Ako názov napovedá, kernel pre algoritmus *AES* používa na vytvorenie hašu funkciu *SHA256*, zatiaľ čo *Blowfish* kernel použije funkciu *SHA1*. Je to z dôvodu, že tieto dve kombinácie algoritmu a hašovacej funkcie používajú jedny z najznámejších voľne dostupných editorov a to LibreOffice[2] a Apache OpenOffice[3].

[2] `https://www.libreoffice.org/`
[3] `https://www.openoffice.org/`

Kapitola 9

Experimenty

V tejto kapitole sú popísane experimenty, ktorých cieľom bolo zmerať rýchlosť implementovaného modulu pri obnove hesiel. Pre meranie bola vytvorená sada testovacích súborov rôznych typov (.odt, .ods, .odp, etc.). Na ich vytvorenie boli použité programy *LibreOffice* verzie 5.2.2.2 pre verziu ODF dokumentov so šifrovaním pomocou algoritmu *AES* a program *Apache OpenOffice* verzie 4.1.3 pre verziu *Blowfish*.

Pred začiatkom merania bolo potrebné zvoliť si abecedu, z ktorej boli jednotlivé generované. Zvolenou abecedou pri všetkých experimentoch bola abeceda obsahujúca 26 znakov, pozostávajúca z malých písmen znakovej sady ASCII. Použité heslo bolo vždy najhoršou možnou variantou pre generátor pri útoku hrubou silou, t.j. pre dĺžku hesla o troch znakoch bolo zvolené heslo "zzz".

Z dôvodu vysokej časovej náročnosti výpočtov nemohli byť namerané všetky možné kombinácie dĺžok hesiel. Niektoré hodnoty museli byť aproximované na základe nameranej rýchlosti prelamovania hesiel daného modulu. Pri výpočte doby potrebnej na obnovu hesiel bol v týchto zdĺhavejších prípadoch použitý skript, ktorý bol poskytnutý vedúcim tejto práce Ing. Radkom Hranickým. Rýchlosť obnovy hesiel bola odvodená na základe nameraných výsledkov z najdlhšieho merania.

Hodnoty pre dlhšie heslá boli aproximované podľa rovnice 9.1, kde a je počet znakov abecedy, k je maximálna dĺžka hesla a r označuje rýchlosť modulu v počte vyskúšaných hesiel za sekundu. Hodnoty vypočítané týmto spôsobom sú v texte vyznačené kurzívou.

$$x = \sum_{i=1}^{k} a^i / r \tag{9.1}$$

9.1 ODF AES

Táto sekcia obsahuje experimentálne merania obnovy hesiel ODF dokumentu, vytvoreného programom LibreOffice. Jedná sa o testovanie rýchlosti dešifrovania funkcie AES. CPU verzia bola testovaná na procesore Intel Core i7-4700MQ 2,40 GHz, zatiaľ čo GPU akcelerovaná verzia prebiehala na stroji s grafickou kartou NVIDIA GeForce GTX 1050 Ti. Hlavným cieľom tohto experimentu bolo overiť predpoklad, že obnova hesiel na GPU, bude podstatne rýchlejšia ako verzia s CPU. Ďalším cieľom bolo porovnať zrýchlenie CPU verzie v závislosti od počtu použitých vlákien. Predpokladom pre tento experiment bolo približne dvojnásobné zrýchlenie pri využití dvojnásobného počtu vlákien. Na záver boli všetky namerané hodnoty porovnané.

	Dĺžka hesla	3	4	5	6	7	8	9	Rýchlosť
Čas	1 vlákno	28s	12m 7s	5h 16m	*5d 17h*	*148d 1h*	*10y 199d*	*274y 74d*	653 p/s
	2 vlákna	19s	7m 14s	2h 46m	*3d 9h*	*87d 21h*	*6y 95d*	*162y 284d*	1100 p/s
	4 vlákna	11s	4m 23s	1h 51m	*2d 2h*	*54d 12h*	*3y 322d*	*100y 340d*	1774 p/s
	8 vlákien	9s	4m 18s	1h 56m	*1d 23h*	*51d 7h*	*3y 239d*	*95y 14d*	1884 p/ş

Tabuľka 9.1: Výsledky merania na CPU, pre algoritmus AES. Hodnoty vyznačené kurzívou boli vypočítané skriptom.

Vyjdrený predpoklad sa ukázal iba ako čiastočne správny, čo dokazujú aj výsledky z tabulky 9.1. Rozdiel medzi jedným a dvomi využitými vláknami sa pohybuje na hranici dvojnásobného zrýchlenia, no pri štyroch a ôsmich vláknach zrýchlenie nie je tak markantné. Štyri vlákna poskytli oproti dvom iba polovičné zrýchlenie, zatiaľ čo osem vlákien znížilo potrebný čas na nájdenie hesla len o jednu pätinu. Tabuľka 9.2 obsahuje hodnoty namerané pre GPU verziu tohto modulu. Na nich je jasne vidieť niekolkonásobné zrýchlenie oproti cpu verzii. V tomto prípade sa predpoklad ukázal ako správny.

Anomáliou pri použití ôsmich vlákien bol výsledok pre heslo o dĺžke päť znakov. V tomto prípade trvalo programu nájsť heslo o päť minút dlhšie ako pri použití iba štyroch vlákien. Dôvodom tohto spomalenia bolo pravdepodobne vyššie vyťaženie testovacieho stroja programami, bežiacimi na pozadí počas doby testovania.

	Dĺžka hesla	3	4	5	6	7	8	9	Rýchlosť
Čas	GPU	1s	12s	4m 50s	2h 5m 49s	*2d 6h*	*59d 42m*	*4y 75d*	42583 p/s

Tabuľka 9.2: Výsledky experimentov na GPU, pre algoritmus AES. Hodnoty vyznačené kurzívou boli vypočítané skriptom.

9.2 ODF Blowfish

Meranie verzie so šifrovacím algoritmom *Blowfish* bolo obdobné ako predchádzajúce merania s algoritmom *AES*. Hodnoty boli namerané na tom istom CPU a GPU ako boli použité pri AES. Heslá súborov boli taktiež totožné. Najprv bola zmeraná rýchlosť dešifrovania na CPU verzii s rozdielnym počtom využitých vlákien. V druhej fáze merania bola odmeraná rýchlosť obnovy hesiel na GPU. Súbory boli vytvorené programom Apache OpenOffice, obsahovali rovnaké dáta a použité heslá boli zhodné s tými pre predhádzajúcu verziu dokumentov. Procesor a grafická karta taktiež ostali nezmenené. Predpokladané bolo podobné zrýchlenie pri použití viacerých vlakien a grafickej karty ako pri verzii AES.

	Dĺžka hesla	3	4	5	6	7	8	9	Rýchlosť
Čas	1 vlákno	15s	6m 8s	2h 46m	*2d 23h*	*77d 8h*	*5y 186d*	*143y 89d*	1250 p/s
	2 vlákna	8s	3m 24s	1h 30m	*1d 15h*	*42d*	*2y 362d*	*77y 285d*	2302 p/s
	4 vlákna	6s	2m 3s	56m 37s	*1d 1h*	*27d 12h*	*1y 350d*	*50y 338d*	3516 p/s
	8 vlákien	4s	1m 59s	55m 18s	*21h 47m*	*23d 14h*	*1y 248d*	*43y 261d*	4096 p/s

Tabuľka 9.3: Výsledky experimentov na CPU pre algoritmus Blowfish. Hodnoty vyznačené kurzívou boli vypočítané skriptom.

Ako je možné vidieť v tabuľke 9.3, predpoklad o zvyšovaní rýchlosti sa čiastočne naplnil. Pri prechode z jedného vlákna na dve sa proces obnovy hesla urýchlil dvojnásobne. Pri pridaní ďalších dvoch vlákien sa toto zrýchlenie už neprejavilo tak efektívne. Pri štyroch vláknach

bola obnova hesiel rýchlejšia iba o jednu tretinu od variantu s dvomi vláknami. Pri ôsmich vláknach sa zmena pri kratších heslách takmer vôbec neprejavila. Pri odhade pre väčšie dĺžky hesiel sa zrýchlenie síce prejavilo, ale iba minimálne. Tabuľka 9.4 obsahuje namerané hodnoty pre GPU verziu tohto modulu. Znovu sa potvrdilo zrýchlenie oproti CPU verzii, no toto zrýchlenie bolo o niečo menšie ako u AES.

	Dĺžka hesla	3	4	5	6	7	8	9	Rýchlosť
Čas	GPU	1s	16s	6m 30s	2h 48m 38s	*2d 15h*	*68d 13h*	*4y 322d*	36672 p/s

Tabuľka 9.4: Výsledky experimentov na GPU. Hodnoty vyznačené kurzívou neboli namerané ale vypočítané.

9.3 Porovnanie s existujúcimi nástrojmi

Nástroj *hashcat* zatiaľ nemá uvedenú podporu pre súbory ODF, preto nebolo možné porovnať nový modul s týmto nástrojom. Ďalšou možnosťou bol komerčný nástroj Advanced Office Password Recovery (AOPR) Professional Edition verzie 6.21 (build 972) od firmy ElcomSoft. Pri porovnávaní rýchlostí modulu ODF a rýchlosti AOPR bol použitý procesor Intel Core i7-4700MQ 2,40 GHz. Pokus o otestovanie tohto nástroja na grafickej karte NVIDIA GeForce GTX 1050 Ti bohužiaľ skončil neúspešne s chybou *internal:thStartAttack:thCreateThreadsData() failed*. Dôvodom tejto chyby bolo pravdepodobne to, že verzia programu je staršia ako grafická karta. V nasledujúcich tabuľkách je možné vidieť porovnanie rýchlostí medzi navrhnutým modulom pre nástroj Fitcrack a nástrojom AOPR. V tabuľke 9.5 sú porovnané dešifrovania dokumentov zašifrovaných algoritmom AES. Tabuľka 9.6 obsahuje hodnoty pre verziu s algoritmom Blowfish.

		Dĺžka hesla	5	6	7	Rýchlosť
Čas	1 vlákno	Fitcrack	5h 16m	*5d 17h*	*148d 1h*	653 p/s
		AOPR	1h 37m 10s	*1d 18h 6m*	*45d 14h 59m*	2119 p/s
	8 vlákien	Fitcrack	1h 56m	*1d 23h*	*51d 7h*	1884 p/s
		AOPR	47m 18s	*20h 30m*	*22d 5h 2m*	4353 p/s
	GPU	Fitcrack	4m 50s	2h 5m 49s	*2d 6h*	42583 p/s
		AOPR	—	—	—	—

Tabuľka 9.5: Porovnanie rýchlostí obnovy hesiel s dokumentov zašifrovaných pomocou AES, medzi novým modulom Fitcracku a AOPR

		Dĺžka hesla	5	6	7	Rýchlosť
Čas	1 vlákno	Fitcrack	2h 46m	*2d 23h*	*77d 8h*	1250 p/s
		AOPR	1h 31m 40s	*1d 15h 44m*	*43d 1h 5m*	2246 p/s
	8 vlákien	Fitcrack	55m 18s	*21h 47m*	*23d 14h*	4096 p/s
		AOPR	47m 26s	*20h 33m*	*22d 6h 30m*	4341 p/s
	GPU	Fitcrack	6m 30s	2h 48m 38s	*2d 15h*	36672 p/s
		AOPR	—	—	—	—

Tabuľka 9.6: Porovnanie rýchlostí obnovy hesiel s dokumentov zašifrovaných pomocou Blowfish, medzi novým modulom Fitcracku a nástrojom AOPR

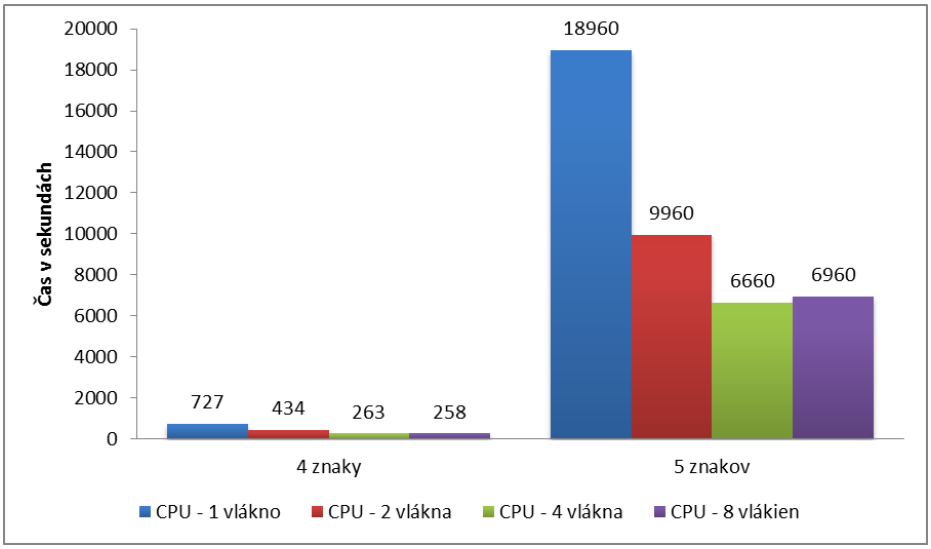
Po porovnaní nameraných hodnôt môžeme vidieť rozdiel v rýchlostiach pre CPU verziu modulu. Pri použití AES je rozdiel podstatne väčší ako pri Blowfish, ktorý svojou rýchlosťou pri ôsmich vláknach AOPR dobieha. Rýchlosť na GPU sa pre nástroj AOPR od ElcomSoftu nepodarilo odmerať z dôvodu nekompatibility s grafickou kartou. V tomto ohľade Fitcrack problém nemal a bez problémov zvládol výpočty aj na jednej z najnovších grafických kariet.

9.4 Zhrnutie

Táto sekcia obsahuje zhrnutie všetkých výsledkov meraní na CPU, GPU a ich porovnanie s nástrojom AOPR od firmy ElcomSoft.

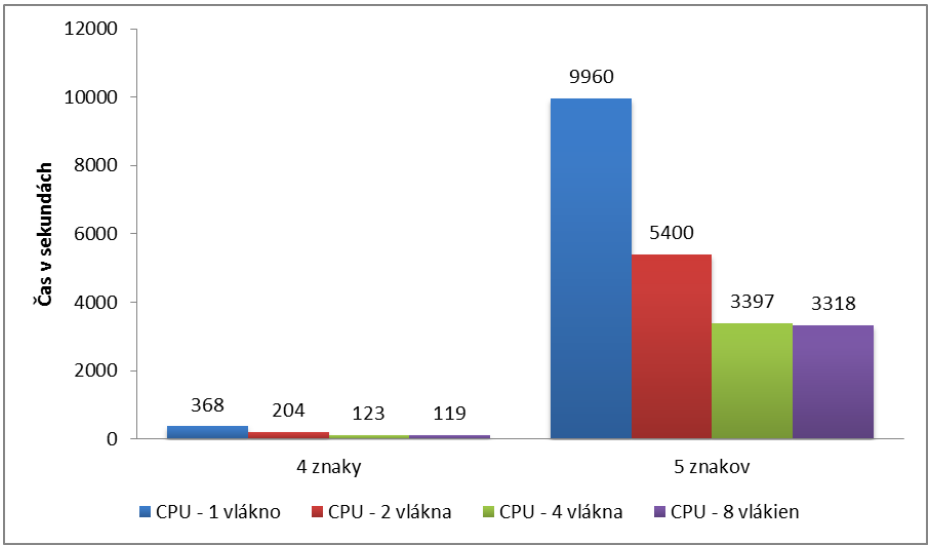
9.4.1 CPU

Merania modulu na CPU, ktorých výsledky možno vidieť na obrázkoch 9.1 a 9.2, poukázali na skoro dvojnásobné zrýchlenie pri použití dvoch vlákien. Použitie štyroch vlákien zaznamenalo zrýchlenie iba polovičné. Najzaujímavejším výsledkom tohoto merania bolo meranie na ôsmich vláknach. Procesor, na ktorom merania prebiehali, má iba štyri fyzické jadrá ale využíva technológiu Hyper Threading[1] na simuláciu ďalších štyroch vlákien. Z výsledkov však je jasné, že toto proces obnovy hesiel urýchlilo minimálne.



Obr. 9.1: Porovnanie času potrebného na obnovu hesla pri zvyšujúcom sa počte použitých vlákien pre verziu s algoritmom *AES*.

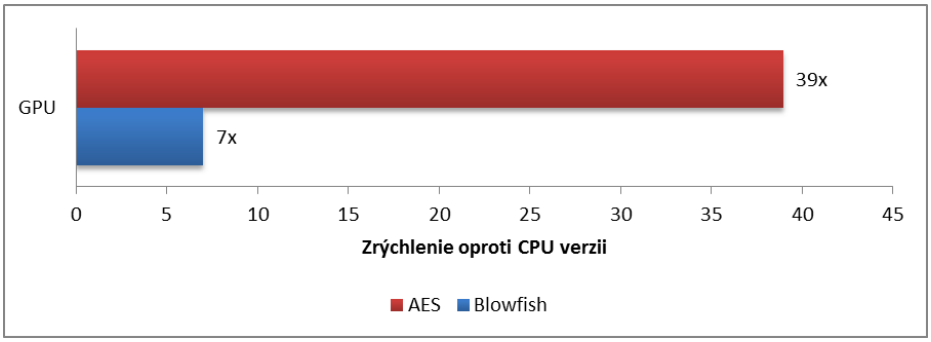
[1] `https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`



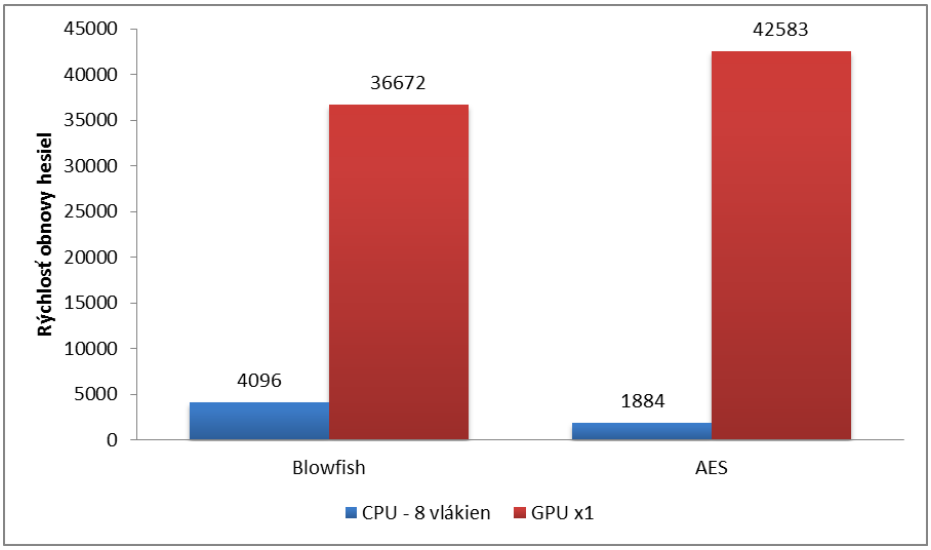
Obr. 9.2: Porovnanie času potrebného na obnovu hesla pri použití rozdielného počtu vlákien procesora pre verziu *Blowfish*.

9.4.2 GPU

Meraniami rýchlosti obnovy hesiel na GPU bolo dokázané viacnásobné zrýchlenie ako sa predpokladalo. Nebolo však dosiahnuté rovnaké zrýchlenie pri oboch verziách modulu. Zrýchlenia sú zobrazené na obrázku 9.3, z ktorého je vidieť, že verzia s algoritmom AES dosiahla skoro šesťkrát vačšie zrýchlenie na GPU ako Blowfish. Na obrázku 9.4 sú graficky znázornené rýchlosti obnovy hesiel oboch verzií. Aj keď bol Blowfish na CPU dvojnásobne rýchlejší ako AES, na GPU rýchlosťou zaostával vďaka optimalizáciám algoritmu AES, ktoré boli implementované skoršími modulmi.



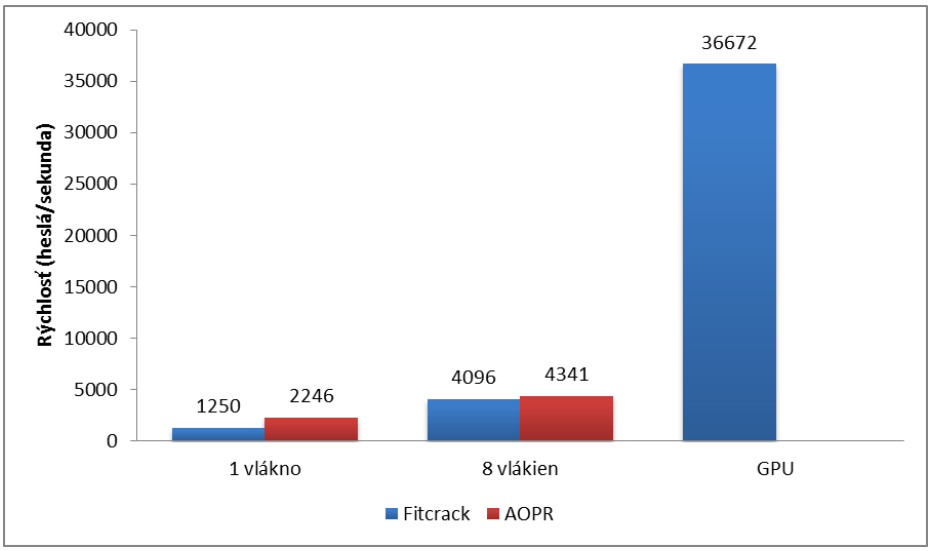
Obr. 9.3: Rozdielne zrýchlenie verzií modulu je zapríčinené nedostatočnou optimalizáciou kernelu pre Blowfish.



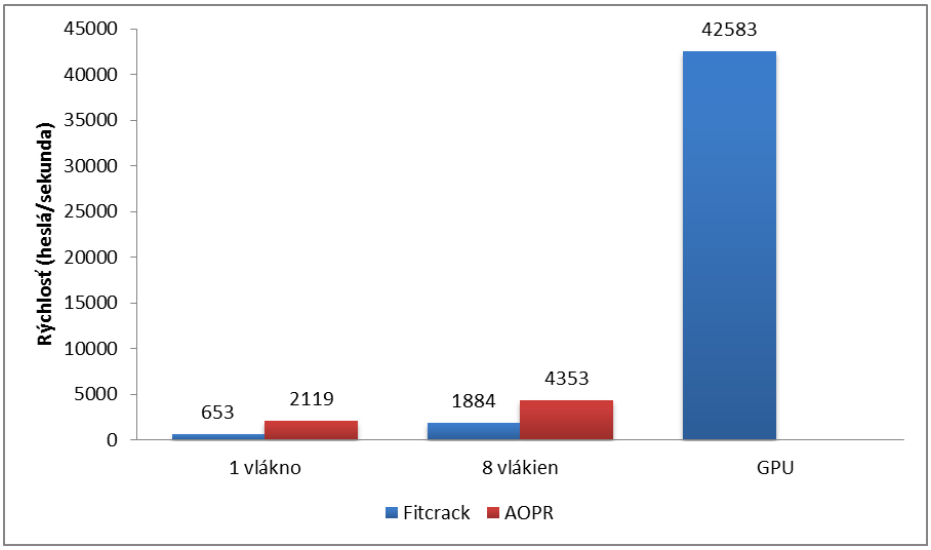
Obr. 9.4: Porovnanie rýchlostí obnovy hesiel pri použití procesora a grafickej karty.

9.4.3 Existujúce nástroje

Obrázky 9.5 a 9.6 znázorňujú hodnoty z predchádzajúcich tabuliek vo forme grafov. Na obrázku 9.5 sú znázornené hodnoty, namerané na navrhnutom module a komerčnom nástroji AOPR od firmy ElcomSoft. Pri porovnaní hodnôt vykazuje navrhnutý modul pri verzii Blowfish menšiu rýchlosť ako AOPR. AES verzia navrhnutého modulu dosahuje iba polovičnú rychlosť ako nástroj AOPR, čo je znázornené grafom na obrázku 9.6. Porovnanie rýchlostí na grafickej karte nebolo možné, ako už bolo spomenuté na začiatku kapitoly 9.3, z dôvodu nekompatibility AOPR s novšou grafickou kartou. Nástroj Fitcrack s grafickou kartou problémy nemal, preto sú namerané hodnoty modulu ODF prítomné v tabuľke aj v grafe.



Obr. 9.5: Porovnanie rýchlosti nástrojov pri Blowfish.



Obr. 9.6: Porovnanie rýchlosti nástrojov pri AES.

Z nameraných výsledkov je vidno badateľný rozdiel rýchlostí pri použití GPU na obnovu hesiel. Pri použití jedného GPU zariadenia sa doba obnovy pre *Blowfish* skrátila sedemkrát oproti CPU. *AES* bol na GPU až tridsaťdeväťkrát rýchlejší oproti výpočtu na CPU. V oboch prípadoch boli použité šesť znakové heslá s využitím ôsmich vlákien na CPU. Tento rozdiel znázorňuje aj obrázok 9.4. Rozdielne zrýchlenie týchto dvoch verzií je zapríčinené optimalizáciou kernelov. Zatiaľ čo kernel pre *AES* používa funkcie optimalizované už predchádzajúcimi modulmi, *Blowfish* kernel nebol optimalizovaný na rovnakej úrovni. Obrázok 9.3 poukazuje na rozdiel medzi zrýchlením oboch verzií modulu ODF na GPU.

Kapitola 10

Záver

Cieľom tejto práce bolo analyzovať šifrovanie súborov formátu Open Document Format (ODF) a navrhnúť modul pre nástroj Fitcrack, ktorý bude schopný obnovy hesiel zo súborov tohto formátu. Po analýze štruktúry súborov ODF boli z príslušných súborov získané informácie, potrebné na overenie hesla. Na základe poznatkov o hašovacích funkciách a šifrovacích algoritmoch, ktoré môžu byť použité, bol navrhnutý nový modul. Pri navrhovaní modulu bolo zistené, že iba programy LibreOffice a Apache OpenOffice umožňujú šifrovanie obsahu súborov. Tieto programy používajú rozdielnu kombináciu hašovacích funkcií a šifrovacích algoritmov. Program LibreOffice používa kombináciu funkcie SHA256 s algoritmom AES v CBC režime a Apache OpenOffice kombináciu funkcie SHA1 s algoritmom Blowfish v CFB režime. Na základe tohto zistenia bola zvolená podpora iba pre tieto funkcie a algoritmy.

Pri následnej implementácii boli vytvorené pre GPU dva varianty modulu ODF na základe algoritmu použitého pri šifrovaní. Prvý variant pre algoritmus AES, druhý pre Blowfish.

Po implementácii bola u oboch verzií modulu zmeraná rýchlosť obnovy hesiel vo verzii CPU a GPU. V ďalšom kroku bol zmeraný vplyv použitia viacerých vlákien procesora na rýchlosť procesu obnovy hesiel. Výsledky boli na záver porovnané s komerčným nástrojom Advanced Office Password Recovery (AOPR) od firmy ElcomSoft. Zrýchlenie pri použití viacerých grafických kariet sa nepodarilo zmerať z dôvodu nefunkčnosti stroja.

Z nameraných výsledkov je možné pozorovať u modulu vo verzii CPU nižšiu rýchlosť obnovy hesiel oproti nástroju AOPR. Modul však funguje bez problémov aj na novších grafických kartách na rozdiel od nástroju AOPR. Tento nedokázal spustiť dešifráciu súboru na grafickej karte, ktorá bola vydaná neskôr ako verzia programu AOPR. Ďalším nedostatkom, ktorý vyplynul z nameraných hodnôt, je výrazne menšie zrýchlenie GPU verzie variantu Blowfish oproti variantu AES.

Pri experimentoch s rôznym počtom použitých vlákien procesora boli namerané zaujímavé výsledky. Predpokladom bolo približne dvojnásobné zrýchlenie pri použití dvojnásobného počtu vlákien. Tento predpoklad sa naplnil pri použití dvoch vlákien a čiastočne pri použití štyroch vlákien. V prípade použitia ôsmich vlákien sa predpoklad nenaplnil. Zrýchlenie bolo minimálne alebo došlo k spomaleniu procesu. Príčinou tejto anomálie je skutočnosť, že procesor použitý pri testovaní mal iba štyri fyzické jadrá. Ďalšie štyri jadrá boli virtuálne pridané pomocou technológie Hyper Threading. Z toho vyplýva, že Hyper Threading nemá na rýchlosť obnovy hesiel vplyv.

Ďalší vývoj modulu teda bude spočívať v jeho optimalizácii pre grafické karty. Prioritne bude potrebné optimalizovať algoritmus Blowfish, aby jeho zrýchlenie dosiahlo úrovne algoritmu AES.

Literatúra

[1] Barker, W. C.; Barker, E. B.: SP 800-67 Rev. 1. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. Technická zpráva, Gaithersburg, MD, United States, 2012, [online]. [cit. 2017-05-09].
URL `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf`

[2] Cover, R.: OASIS Open Document Format for Office Applications (OpenDocument) Version 1.2-Part 3: Packages. 2003, [online]. [cit. 2017-01-09].
URL `http://docs.oasis-open.org/office/v1.2/cs01/OpenDocument-v1.2-cs01-part3.html`

[3] Dobbertin, H.; Bosselaers, A.; Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In *FSE*, *LNCS*, ročník 1039, editace D. Gollmann, Springer, 1996, ISBN 3-540-60865-6, s. 71–82, [online]. [cit. 2017-01-12].
URL
`http://homes.esat.kuleuven.be/~bosselae/ripemd160/pdf/AB-9601/AB-9601.pdf`

[4] FIPS, P.: 46-3: Data Encryption Standard (DES). *National Institute of Standards and Technology*, ročník 25, č. 10, 1999: s. 1–22, [online]. [cit. 2017-01-13].
URL `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`

[5] FIPS, P.: 197. Advanced Encryption Standard (AES), National Institute of Standards and Technology, US Department of Commerce. 2001, [online]. [cit. 2017-01-12].
URL `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

[6] FIPS, P.: 180-4-Federal Information Processing Standards Publication-Secure Hash Standard (SHS)-National Institute of Standards and Technology Gaithersburg. 2012, [online]. [cit. 2017-05-14].
URL `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`

[7] Howes, L.; Munshi, A.: *The OpenCL Specification*. [online]. [cit. 2016-12-18].
URL `https://www.khronos.org/registry/cl/specs/opencl-2.1.pdf`

[8] Hranický, R.; Holkovič, M.; Matoušek, P.; aj.: On Efficiency of Distributed Password Recovery. *The Journal of Digital Forensics, Security and Law*, ročník 11, č. 2, 2016: s. 79–96, ISSN 1558-7215, [online]. [cit. 2017-01-13].
URL `http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11276`

[9] Menezes, A. J.; Vanstone, S. A.; Oorschot, P. C. V.: *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, Inc., první vydání, 1996, ISBN 0849385237.

[10] Munshi, A.; Gaster, B.; Mattson, T. G.; aj.: *OpenCL programming guide*. Pearson Education, 2011.

[11] Schmied, J.: *GPU akcelerované prolamování šifer*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2013, [online]. [cit. 2016-12-28]. URL `http://www.fit.vutbr.cz/study/DP/DP.php?id=16858`

[12] Schneier, B.: *Description of a new variable-length key, 64-bit block cipher (Blowfish)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, ISBN 978-3-540-48456-1, s. 191–204, doi:10.1007/3-540-58108-1_24, [online]. [cit. 2017-05-14]. URL `http://dx.doi.org/10.1007/3-540-58108-1_24`

[13] Tompson, J.; Schlachter, K.: An introduction to the opencl programming model. *Person Education*, ročník 49, 2012, [online]. [cit. 2017-05-12]. URL `http://cims.nyu.edu/~schlacht/OpenCLModel.pdf`

Príloha A

Obsah CD

latex/ – zdrojové súbory technickej správy v jazyku LaTeX

src/ – zdrojové súbory nástroja Fitcrack

test/ – súbory pre testovanie implementovaného rozšírenia

projekt.pdf – technická správa v PDF dokumente

README – súbor s informáciami o vytvorených a upravených súboroch