VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OBNOVA HESEL ARCHIVŮ RAR, BZIP A GZIP S VYUŽITÍM GPU

PASSWORD RECOVERY OF RAR, BZIP, AND GZIP ARCHIVES USING GPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE DÁVID MIKUŠ
AUTHOR

VEDOUCÍ PRÁCE Ing. RADEK HRANICKÝ
SUPERVISOR

BRNO 2016

Zadání bakalářské práce/18740/2015/xmikus15

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Mikuš Dávid**
Obor: Informační technologie
Téma: **Obnova hesel archivů RAR, BZIP a GZIP s využitím GPU**
Password Recovery of RAR, BZIP, and GZIP Archives Using GPU
Kategorie: Bezpečnost

Pokyny:

1. Seznamte se s architekturou a implementací nástroje Wrathion.
2. Nastudujte techniky šifrování archivů RAR a možnosti šifrování archivů GZIP/BZIP pomocí OpenSSL.
3. Navrhněte sadu modulů pro nástroj Wrathion, které doplní jeho functionalitu o podporu formátu RAR a formátů GZIP/BZIP šifrovaných pomocí OpenSSL.
4. Sadu modulů implementujte (včetně kódu pro akceleraci pomocí GPU).
5. Proveďte měření výkonu navržených modulů a výsledky porovnejte s konkurenčními nástroji.
6. Zhodnoťte dosažené výsledky.

Literatura:

- HU, Guang, MA, Jianhua, HUANG, Benxiong. Password Recovery for RAR Files Using CUDA. In: *DASC '09. Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing.* Chengdu: IEEE *2009. s. 486-490. ISBN* 978-0-7695-3929.
- 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 2015
Datum odevzdání: 18. května 2016

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

Táto práca sa zaoberá popisom kryptografického zabezpečenia a implementácie obnovy hesla súborov vo formátoch RAR, GZIP, BZIP. Na obnovu hesla je použitý multi platformový nástroj Wrathion, ktorý dokáže pomocou modulov a generátorom hesiel získať heslo zo zašifrovaného súboru. Modul zabezpečuje prečítanie súboru a uloženie dát, ktoré sa ďalej použijú pri overení platnosti hesla. Na akceleráciu výpočtov se využíva grafická karta pomocou frameworku OpenCL. Jadrom práce je implementácia modulov pre dané formáty v nástroji Wrathion a následne meranie rýchlosti a porovnávanie s konkurenčnými nástrojmi.

Abstract

This thesis describes cryptographic protection and the implementation of file password of RAR, GZIP, BZIP formats. For password recovery multi platform Wrathion tool is used, the tool can obtain a password from an encrypted file by using its modules and password generators. The module provides parsing of input file and storing data, which are later used for password validation. To accelerate calculations, graphics card and OpenCL framework is used. The core of thesis is the implementation of Wrathion's modules for password recovery of the formats denoted above and measuring the speed and comparision with competeting tools.

Kľúčové slová

Wrathion, OpenCL, RAR, GZIP, BZIP, OpenSSL, GPU, heslo, obnova

Keywords

Wrathion, OpenCL, RAR, GZIP, BZIP, OpenSSL, GPU, password, recovery

Citácia

MIKUŠ, Dávid. *Obnova hesel archivů RAR, BZIP a GZIP s využitím GPU.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Hranický Radek.

Obnova hesel archivů RAR, BZIP a GZIP s využitím GPU

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Hranického. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Dávid Mikuš
15. mája 2016

Poďakovanie

Rád by som poďakoval svojmu veducému Ing. Radkovi Hranickému za odborné vedenia a pomoc pri písani tejto práce a Ing. Martinovi Holkovičovi za prípravu testovacieho serveru a rady k písaniu tejto práci.

© Dávid Mikuš, 2016.
Táto práca vznikla ako školské dielo na FIT VUT v Brně. Práca je chránená autorským zákonom a jej využitie bez poskytnutia oprávnenia autorom je nezákonné, s výnimkou zákonne definovaných prípadov.

Obsah

1 Úvod **3**

2 OpenCL **4**
2.1 Platformový model 4
2.2 Pamäťový model 5
2.3 Exekučný model 6
2.4 Programovací model 6

3 Wrathion **7**
3.1 Architektúra 7
3.2 Obnova hesla pomocou CPU 8
3.3 Obnova hesla pomocou GPU 9

4 Kryptografické algoritmy **12**
4.1 Šifrovacie funkcie 12
4.1.1 AES 12
4.1.2 Blokové šifry 14
4.2 Hašovacie funkcie 15
4.2.1 MD5 15
4.2.2 SHA-1 16
4.2.3 SHA-256 17

5 Rozbor formátov **18**
5.1 RAR 18
5.1.1 RAR 5.0 18
5.1.2 RAR 3.0 19
5.2 GZIP 20
5.3 BZIP2 21

6 OpenSSL **22**

7 Návrh modulov **23**
7.1 RAR 5.0 23
7.2 RAR 3.0 25
7.3 OpenSSL 26

8 Implementácia **29**
8.1 RAR . . . 29
8.1.1 RAR 5 . . . 29
8.1.2 RAR 3 . . . 30
8.2 OpenSSL . . . 30
8.3 Integrácia . . . 31

9 Experimenty a testovanie **33**
9.1 Testovacia zostava . . . 33
9.2 OpenSSL . . . 33
9.2.1 CPU Škálovateľnosť . . . 33
9.2.2 GPU Škálovateľnosť . . . 34
9.2.3 Porovnanie dĺžky kľúča . . . 35
9.2.4 Zhrnutie . . . 36
9.3 RAR . . . 36
9.3.1 RAR 5 . . . 36
9.3.2 RAR 3 . . . 37
9.3.3 Porovnanie nástrojov . . . 38

10 Záver **40**

Literatúra **41**

Prílohy **42**

A Obsah CD **43**

Kapitola 1

Úvod

Od počiatku ľudstva, ľudia medzi sebou potrebovali komunikovať, vymieňali si medzi sebou informácie rôznymi spôsobmi. Samotná informácia mohla byť veľmi významná a bolo nutné aby obe zúčastnené strany mohli komunikovať čo najrýchlejšie ,ale aj bezpečne, teda aby danú informáciu vedela pochopiť len určená strana.

Aktuálne najrýchlejšiu komunikáciu na veľké vzdialenosti nám umožňujú počítače prostredníctvom internetu. Existuje ale veľké riziko, že ku správe sa dostane neoprávnená osoba. Týmto problémom sa zaoberá kryptografia, ktorá rieši šifrovanie dát a prevádza správu do nečitateľnej formy. Táto nečitateľná forma sa dá dešifrovať a teda previesť do čitateľnej formy len pomocou znalosti ,ako napríklad použitá šifra a použité parametre pri samotnom šifrovaní, obvykle heslo.

Medzi najjednoduchšími šiframi boli substitučné metódy, ktoré nahrádzali znak iným znakom, podľa určitého pravidla, napríklad Caesarova šifra[1]. Ďalšia veľmi známa je metóda založená na posune písmen, kde sa znak posunul o určitý počet v rámci abecedy. Toto sú veľmi jednoduché šifry používané v minulosti, odvtedy kryptografia spravila veľký pokrok. Vznikla moderná kryptografia, ktorá bola prispôsobená modernej výpočtovej techniky aby bola bezpečná.

Pre obnovu hesla bol vyvinutý na fakulte FIT multiplatformový nástroj Wrathion, ktorý dokáže obnoviť heslo z určitých formátov súboru. O každý formát sa stará samotný modul a nástroj je teda ľahko rozšíriteľný implementáciu ďalších modulov. Táto práca sa bude zaoberať implementáciou modulov pre formát RAR a aj pre formáty GZIP, BZIP2 ,kde sa použije generické šifrovanie pomocou OpenSSL.

V kapitole 2 bude vysvetlený framework OpenCL, ktorý slúži na akceleráciu výpočtov na grafickej karte. V kapitole 3 sa zameriame na samotný nástroj Wrathion, jeho architektúru a čo zatiaľ dokáže. V kapitole 4 sa pozrieme na šifrovacie algoritmy ktoré sa používajú vo formátoch súborov v ktorých budeme obnovovať heslo. V kapitole 5 si rozoberieme každý formát, aké informácie obsahuje a čo z toho môžeme použiť pre samotnú obnovu hesla. V kapitole 6 stručne popíšeme knižnicu OpenSSL, ktorá sa používa aj pre šifrovanie samotných súborov. V kapitole 7 bude popísane ako získať potrebné informácie zo súborov pre obnovu hesla a následná kontrola hesla. V kapitole 8 rozoberieme samotnú implementáciu modulov. V kapitole 9 porovnáme CPU oproti GPU variante a výkon neimplementovaných modulov vo Wrathion oproti konkurenčným nástrojom.

[1] Zdroj: https://en.wikipedia.org/wiki/Caesar_cipher

Kapitola 2

OpenCL

OpenCL [8] (Open computing language) je multiplatformový štandard pre programovanie heterogénnych systémov pozostávajúcich okrem iných najmä z:

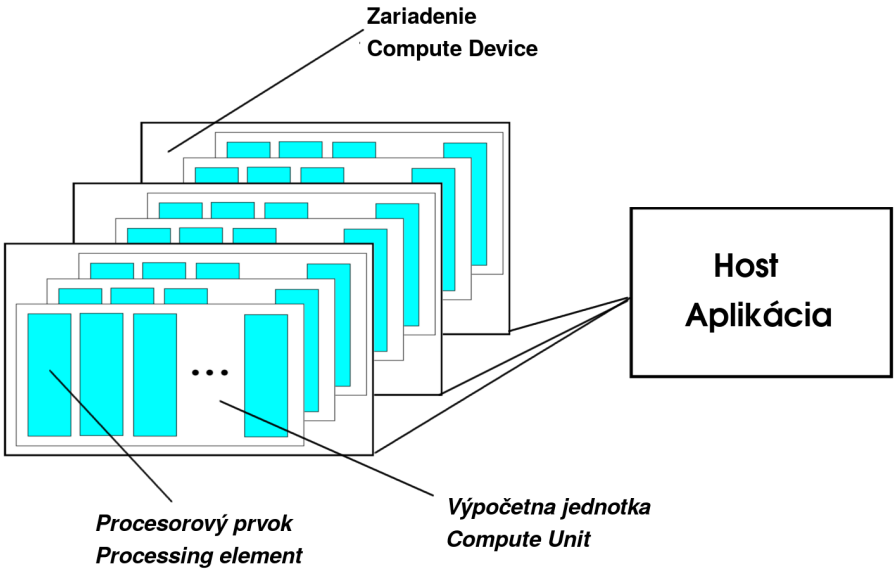
- CPU,
- GPU,
- FPGA.

OpenCL používa jazyk OpenCL C, ktora je podmnožinou C99. Od verzie 2.1 je OpenCL C nahradený aj keď stále podporovaný, jazykom OpenCL C++ ktorý je podmnožinou C++14. Pomocou tohto jazyka sa píšu špecifické funkcie, ktoré sa vykonajú na OpenCL zariadeniach nazývané kernel. Kernely sú prekladané až za behu programu, čo umožňuje aplikácii byť prenosnou medzi rôznymi implementáciami pre rozličné zariadenia.
OpenCL poskytuje nízko úrovňovú abstrakciu hardwaru a framework umožňujúci programovanie.
Jadro OpenCL pozostáva zo 4 modelov:

- Platformový model,
- pamäťový model,
- exekučný model,
- programovací model.

2.1 Platformový model

Model pozostáva z *hosťa* pripojeného k jednému alebo viac OpenCL zariadení. Zariadenie je rozdelené do výpočetnych jednotiek *compute units* (CUs), ktoré sa ďalej delia do procesorových prvkov *processing units* (PUs).



Obr. 2.1: OpenCL - platformový model

OpenCL aplikácia beží na hosťovi a posiela príkazy na vykonanie výpočtu do pracujúcich prvkov (PUs).

OpenCL je navrhnutý pre podporu rozličných zariadení pod jednou platformou. Sú 3 identifikátory verzii OpenCL systému

- Verzia platformy - zahrňuje API ktoré, hosť môže používať s OpenCL(kontext, pamäťové objekty ...)
- Verzia zariadenia - zahrňuje schopnosti zariadenia ako limit zdrojov alebo rozšírenú funkcionalitu
- Verzie OpenCL jazyka - predstavuje programovací jazyk, ktorý vývojár môže použiť. OpenCL jazyk je navrhnutý aby bol spätne kompatibilný, takže stačí ak zariadenia podporuje jeden jazyk.

2.2 Pamäťový model

Zariadenia obsahuje 4 typy pamäti:

1. Globálna pamäť - Každá inštancia kernelu ma prístup k tomuto typu, inštancia môže čítať ale aj zapisovať do akéhokoľvek prvku v tejto pamäti.
2. Pamäť konštánt - Hosť alokuje a inicializuje objekty. Počas vykonávania kernelu je táto oblasť konštantná a dá sa z nej len čítať.
3. Lokálna pamäť - Do tejto pamäti môžu pristupovať len inštancie kernelu z rovnakej pracovnej skupiny. Obsah môžu čítať aj zapisovať.
4. Privátna pamäť - Dostupná len konkrétnej inštancii kernelu. Premenné definované v tejto pamäti nie sú viditeľne inej inštancii. Povolené čítanie aj zápis.

Zariadenie nemôže dynamicky alokovať pamäť, alokáciu môže vykonať len aplikácia v globálnej pamäti alebo lokálnej. Pamäť aplikácie a pamäť zariadenia sú oddelené od seba. Interakcia medzi týmito dvoma pamätiam môže prebehnúť dvoma spôsobmi:

- Explicitným kopírovaním - je nutné vytvoriť úlohu, ktorá skopíruje pamäť.
- Mapovaním regiónov pamäti objektu - Aplikácia môže namapovať region z pamäti objektu zariadenia do svojho adresového priestoru.

2.3 Exekučný model

Vykonávanie OpenCL programu pozostáva z 2 častí:

- Aplikácia - beží na CPU.
- Kernel - beží na paralelnom zariadení (GPU).

Zariadenie obsahuje inštancie kernelu, ktoré sú zoskupované do pracovných skupín. Inštancie v rovnakej pracovnej skupine môžu komunikovať medzi sebou cez lokálnu pamäť. Kernely ktoré sa majú vykonať sa radia do rady a sú vykonávane buď *in-order* alebo *out-order*. *In-order* úloha sa vykoná akonáhle predchádzajúca úloha skončí. *Out-order* úloha sa vykoná hneď ako sú dostupné zdroje zariadenia, je nutná synchronizácia.

2.4 Programovací model

Exekučný model OpenCL podporuje dátový a aj úlohový paralelizmus.
Data sú v pamäti zariadenia a každá inštancia kernelu ma pridelené svoje číslo a s tým spojenú aj svoju časť dát.
V úlohovom paralelizme není možná komunikácia medzi inštanciami kernelu v priebehu vykonávania.

Kapitola 3

Wrathion

Wrathion [11] je multiplatformový nástroj, ktorý umožňuje obnovu hesiel rôznych súborov. Medzi aktuálne podporované formáty patria:

- DOC - Textový formát od spoločnosti Microsoft.
- PDF - Formát na ukladanie dokumentov nezávisle na zariadení na ktorom bolo vytvorené, zabezpečuje aby sa dokument na každom zariadení zobrazil rovnako.
- ZIP - Archivačný formát na pre bezstratovú kompresiu dát.

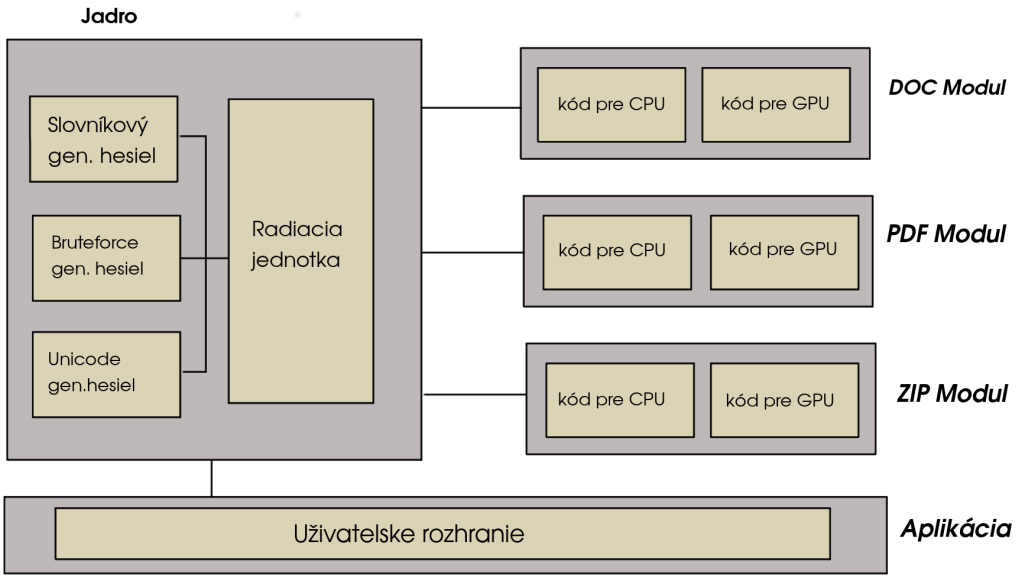
Nástroj je navrhnutý tak aby bol ľahko rozširitelny o ďalšie formáty a ich verzii pomocou modulov. Nástroj obsahuje generátor hesiel ktorý generuje a dodáva heslá, heslo je následne testované či je správne.

- Brute-force - generuje všetky možne kombinácie hesiel zo zadanej abecedy.
- Dictionary - slovníkový, negeneruje, len predáva heslá zo súboru.
- Unicode - zovšeobecnenie brute-force generátoru, znaky sú definované hexadecimálne v Unicode notácii.

Nástroj pre svoje výpočty dokáže využívať procesor (CPU) a taktiež akcelerovať rýchlosť výpočtov pomocou grafickej karty (GPU). Jadro nástroja pre využitie GPU je framework OpenCL (viz kapitola 2) ktorý umožňuje využitie výkonu GPU aj mimo grafických aplikácii a paralelizovať výpočty na svojich jadrách, v prípade Wrathionu ide o paralelizovanie generovania a testovania hesiel.

3.1 Architektúra

Užívateľské rozhranie a teda komunikácia s nástrojom prebieha cez jednoduché CLI. Kontext riadenia sa následne predá jadru aplikácie, ktoré zabezpečuje funkcionalitu celého crackovania. Jadro obsahuje riadiacu jednotku. Ten s pomocou modulov automaticky deteguje formát a šifrovanie. Následne sa načíta modul pre daný formát a spustí sa obnova hesla. Každý modul formátu obsahuje 2 kódy a to pre CPU a GPU pre crackovanie špecifického formátu. Aplikácia štandardne vyberie GPU verziu ak je dostupná a užívateľ nechce explicitne CPU verziu. Podporované je aj viacero CPU i GPU súčasne.

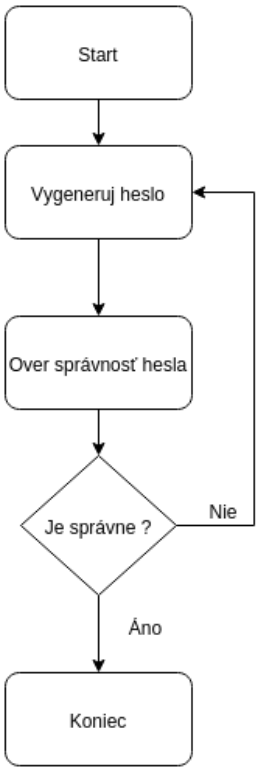


Obr. 3.1: Wrathion - Architektúra [5]

Pre obnovu hesla treba vygenerovať heslo ktoré ma byť otestované. Wrathion obsahuje generátor hesiel ktorý dodáva hesla na otestovanie.

3.2 Obnova hesla pomocou CPU

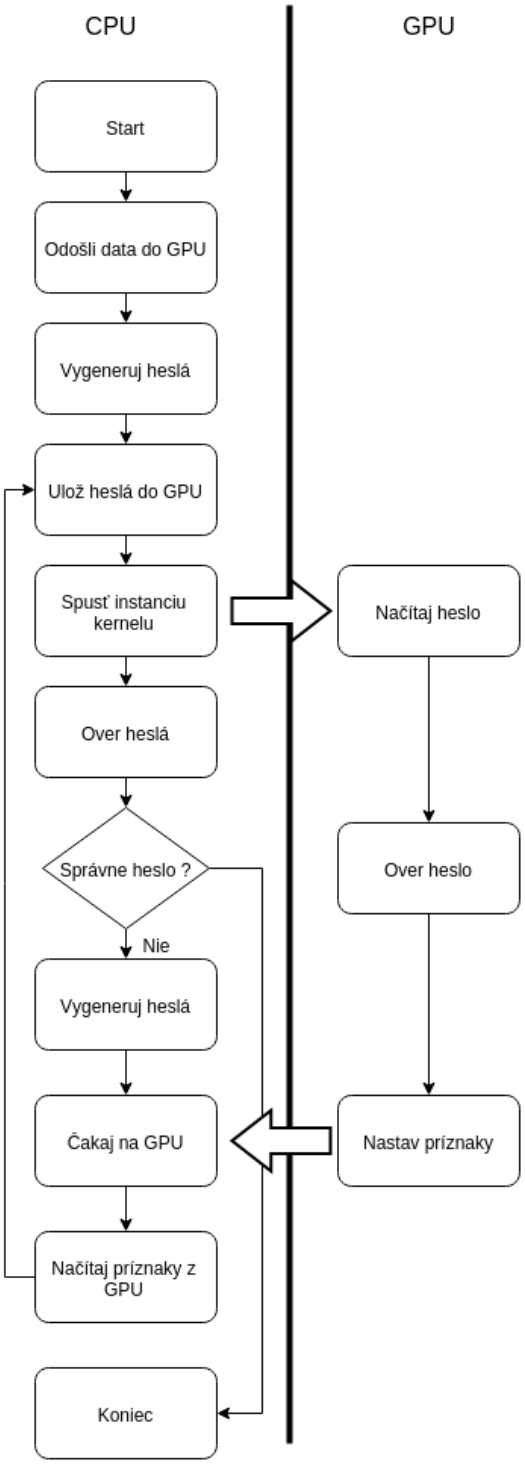
Obnova hesla pomocou CPU varianty je pomerne jednoduchá, stačí vygenerovať heslo a následne ho overiť. Ak je vygenerované heslo zlé, tak sa vygeneruje ďalšie a otestuje. Tento proces sa bude opakovať dokým nenájde správne heslo alebo sa nevyčerpajú heslá z generátora.



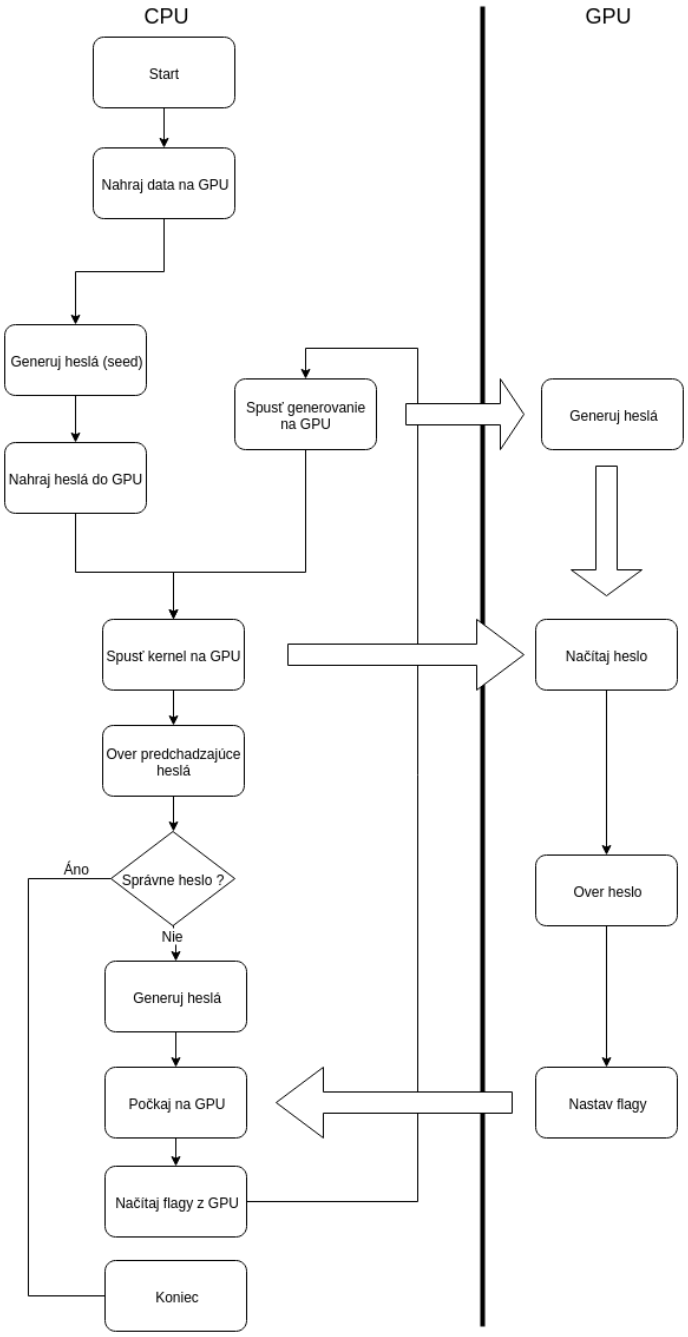
Obr. 3.2: Wrathion - CPU varianta obnovy hesla[11]

3.3 Obnova hesla pomocou GPU

Obnova hesla pomocou GPU je zložitejšie. Je nutné použiť framework OpenCL, teda ho inicializovať, načítať a preložiť OpenCL kód. Následne je nutné do GPU odoslať dáta. Každá inštancia kernelu overuje v jeden moment jedno heslo. Pamäť pre heslo je daná ako maximálna dĺžka hesla a jeden bajt na začiatok, ktorý obsahuje dĺžku uloženého hesla. Teda každá inštancia kernelu obdrží heslo, skontroluje ho a výsledok predá CPU.



Obr. 3.3: Wrathion - GPU varianta obnovy hesla s generátorom hesiel v CPU[11]



Obr. 3.4: Wrathion - GPU varianta obnovy hesla s generátorom hesiel v GPU[11]

Kapitola 4

Kryptografické algoritmy

Zmyslom kryptografie je utajiť obsah správy prevodom do podoby, ktorá je čitateľná len so špecializovanými vedomosťami.
Kryptografické algoritmy prevádzajú čistý text na nečitateľnú formu, nazývanú šifrovaný text. Používajú kľúč, ktorý sa používa na šifrovanie a dešifrovanie textu.
Kryptografia sa delí na:

- Symetrickú - používa rovnaký kľúč na šifrovanie aj na dešifrovanie.
- Asymetrickú - používa verejný kľúč na šifrovanie a súkromný pre dešifrovanie.

4.1 Šifrovacie funkcie

V tejto kapitole rozoberieme šifrovaciu funkciu AES, ako funguje a čo vykonáva. A ďalej blokové šifry ktoré sa používajú spolu so šifrovacími funkciami.

4.1.1 AES

Advanced Encryption Standard [1] je založený na algoritmu *Rijndael*. Je to algoritmus používaný k šifrovaniu dát. Ide o symetrickú blokovú šifru, takže používa rovnaký kľúč aj pre šifrovanie aj dešifrovanie. Data sú rozdelené do blokov veľkosti 128 bitov. Veľkosť kľúča je 128, 192 alebo 256 bitov. AES pracuje s maticou o veľkosti 4x4. Dĺžka použitého kľúča určuje koľko kôl sa má vykonať:

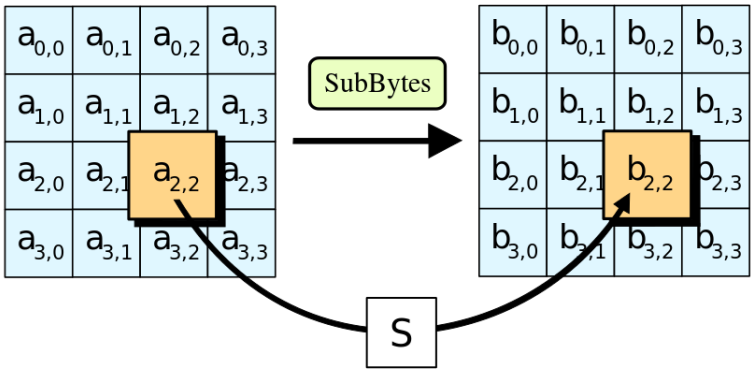
- 128-bit - 10 kôl,
- 192-bit - 12 kôl,
- 256-bit - 14 kôl.

Proces šifrovania sa skladá z 4 častí:

1. Expanzia kľúča - pod-kľúče sú odvodené z kľúča šifry použitím *Rijndael key schedule*
2. Inicializačná časť - urobenie kroku *Add Round Key* - XOR pod-kľúča s blokom
3. Iterácia - vykonávanie jednotlivých kôl.
4. Záverečná časť - vykonanie kola bez kroku *MixColumns* - kombinovanie stĺpcov

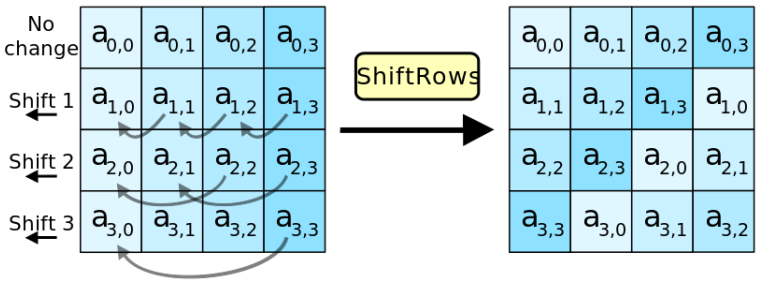
Každé kolo iterácie obsahuje 4 kroky:

1. Zámena bajtov - *SubBytes* - Každý bajt v matici je nahradený pomocou *S-Boxu*. Tento krok zabezpečuje nelinearitu v šifre.



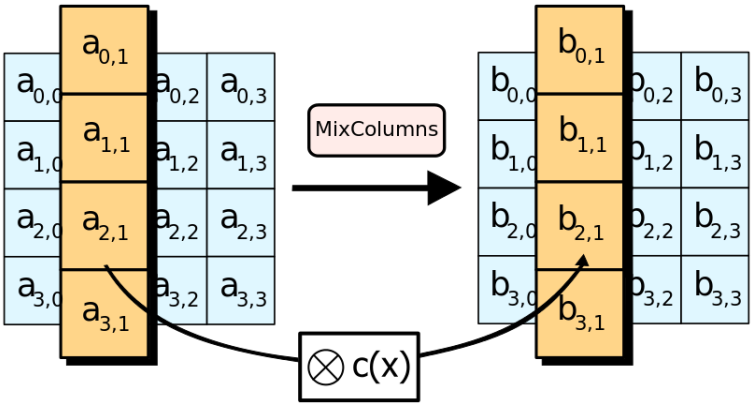
Obr. 4.1: AES - Zámena riadkov[1]

2. Posun riadkov - *ShiftRows* - Cyklický posun riadkov, každý riadok je rotovaný doľava, *i-tý riadok rotuje o i-1 doľava.*



Obr. 4.2: AES - Posun riadkov[1]

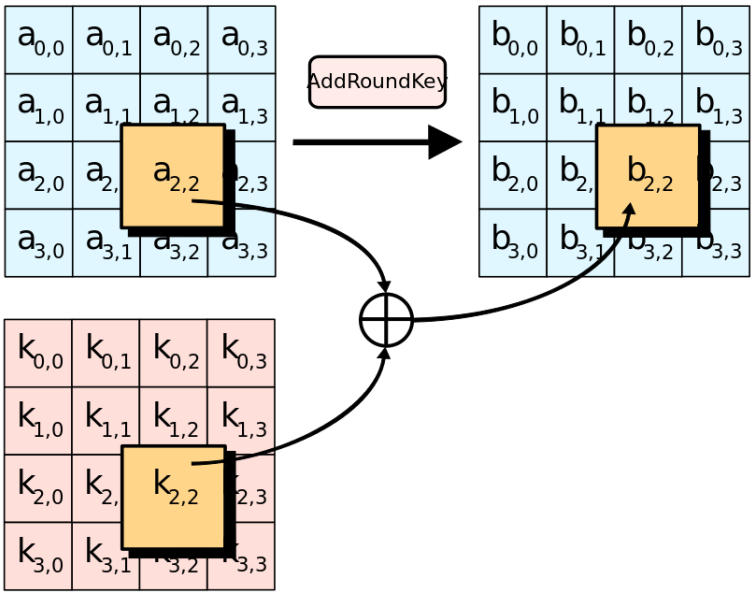
3. *Kombinovanie stĺpcov - MixColumns - Každý stĺpec je vynásobený maticou M v GF(2^8) - Galoisove pole*



Obr. 4.3: AES - Kombinovanie stĺpcov[1]

[1]Zdroj: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

4. Pridanie pod-kľúča - AddRoundKey - vstupná matica sa XORuje s pod-kľúčom.



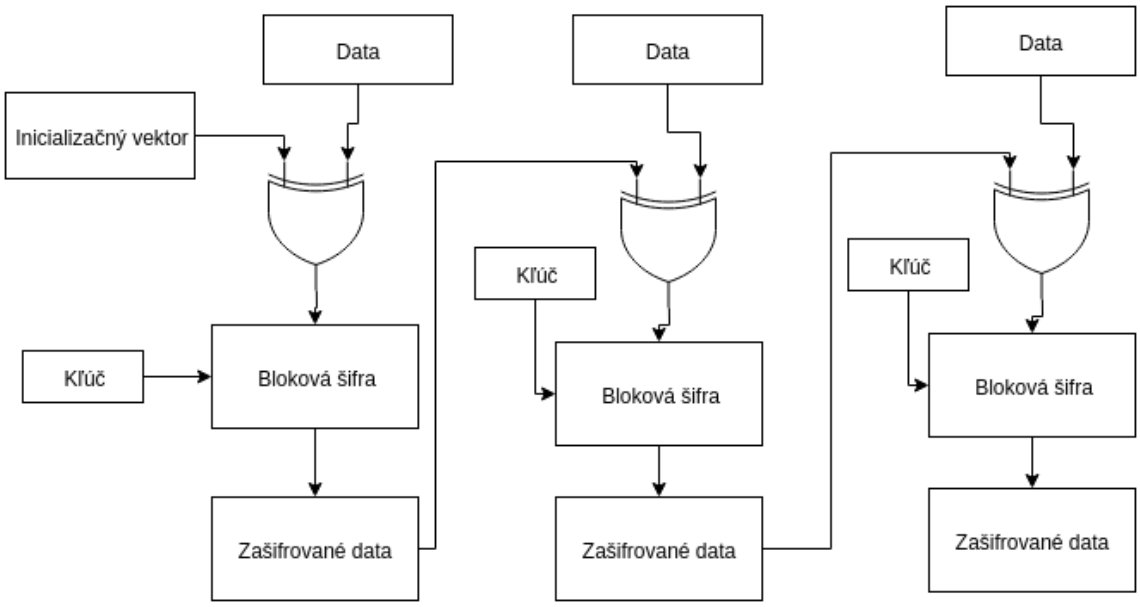
Obr. 4.4: AES - Pridanie pod-kľúča[1]

4.1.2 Blokové šifry

Blokové šifra je typ symetrickej šifry, ktorá pracuje s blokmi pevnej dĺžky. Data rozdeľuje do viacero blokov a do zvyšného miesta je pridaná výplň. Pri šifrovaní sa každý blok zašifruje kľúčom pomocou šifrovacieho algoritmu. Bezpečne šifrovanie je vhodné len pre šifrovanie jedného bloku. Pre bezpečnejšie šifrovanie viacero blokov boli navrhnuté režimy ktoré pridajú do šifrovania ďalší vstup:

- ECB - Electronic Codebook
- OFB - Output Feedback
- CFB - Cipher Feedback
- CBC - Cipher Block Chaining

CBC používa AES. Každý blok vstupu pred jeho samotným šifrovaním je XOR-ovaný s predchádzajúcim blokom zašifrovaného text. Tým pádom každý blok závisí na všetkých predchádzajúcich blokoch ktoré boli spracované. Pre prvý blok sa používa inicializačný vektor.



Obr. 4.5: Princíp režimu blokovej šifry CBC

4.2 Hašovacie funkcie

Hašovacia funkcia je matematická funkcia pre prevod vstupného reťazca na kratší výstupný reťazec označovaný ako haš (angl. *hash*) alebo aj odtlačok vstupných dát. Pre rovnaký vstup je vždy rovnaký výstup.

Kryptografická hašovacia funkcia kladie dôraz na bezpečnosť a používa sa overenie integrity dát alebo autentifikáciu. Takáto funkcia nemôže mat k sebe inverznú funkciu, znalosť výstupu nevedie k znalosti pôvodných vstupných dát. Tieto funkcie sa používajú aj na odvodenie potrebných dát pre kryptografické algoritmy, kde je nutné vykonanie veľkého množstva iterácii funkcie čo zvyšuje potrebný čas na útok hrubou silou.

Medzi tieto funkcie patrí:

- MD5 - Message Digest
- SHA-1 - Secure Hash Algorithm 1
- SHA-256 - Secure Hash Algorithm 256

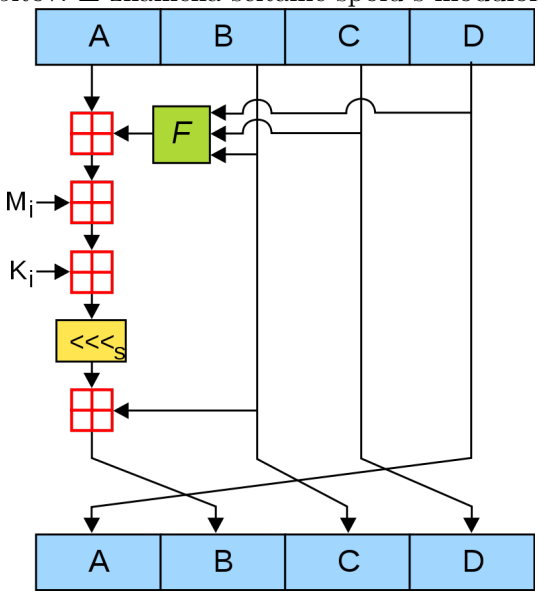
4.2.1 MD5

MD5 (Message Digest) [7] je nástupcom MD4, navrhnutý v roku 1991. Jedná sa o kryptografickú hašovaciu funkciu, ktorá produkuje 128 bitový výstup.

Vstup je rozdelený na 512 bitové bloky a vyplnený tak, aby jeho dĺžka bola deliteľná číslom 512. Za vstup sa pridá jeden bit '1' a následne sa dopĺňajú nulové bity, pokiaľ dĺžka vstupu modulu 512 není 448. Do zvyšných 64 bitov sa uloží pôvodná dĺžka vstupu.

Algoritmus pracuje so 128 bitmi, ktoré sú rozdelené po 32 bitov do 4 blokov A-D. Každý tento blok je inicializovaný vlastnou konštantou. Spracováva každý 512 bitový blok samostatne, ale výsledok predchádzajúceho bloku je nutné poznať pred spracovaním ďalšieho. V 64 kolách sa aplikuje kompresná funkcia, výsledok je pripočítaný ku vnútorným blokom A-D.

Na obrázku 4.2.1 je znázornene jedno kolo MD5. Pozostáva zo 64 takýchto operácii zoskupených do 4 skupín. F je kompresná funkcia, pre každú skupinu rozdielna. M_i značí 32 bitový blok vstupu a K_i 32 bitovú konštantu, rozdielnu pre každú operáciu. $\langle\langle\langle_s$ značí bitový posun vľavo o `s` bitov. $\boxplus$ znamená sčítanie spolu s modulom 2^{32}.



Obr. 4.6: Jedno kolo algoritmu MD5 [2]

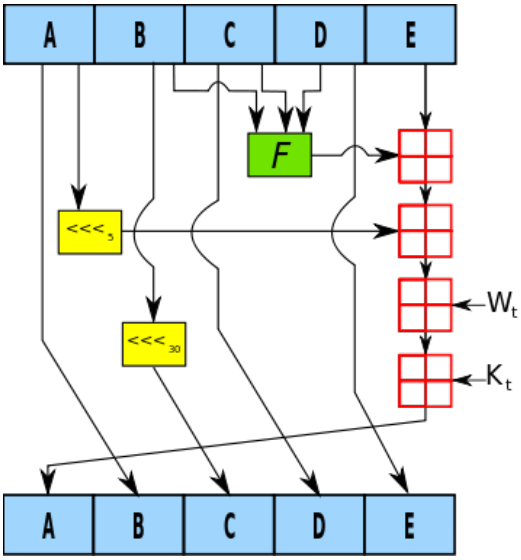
4.2.2 SHA-1

SHA-1 (Secure Hash Algorithm 1) [4] je založená na princípoch MD5 (viz podsekcia 4.2.1). Tento algoritmus bol publikovaný v roku 1993 a následne po zmene v kompresnej funkcii schválený v roku 1995.

Zarovnanie je rovnaké ako u MD5. Líši sa veľkosťou vnútorného stavu, ktorý je 160 bitový a teda produkuje aj výstup v takejto dĺžke. Ďalej obsahuje rozdielnu kompresnú funkciu, inicializačné vektory, konštanty a počet kôl. SHA-1 vykonáva 80 kôl.

Na obrázku 4.2.2 je znázornene jedno kolo algoritmu. A-E sú 32 bitové stavy. F je kompresná funkcia. $\langle\langle\langle_n$ značí bitový posun vľavo o `n` bitov kde `n` sa líši pre každú operáciu. W_t je rozšírený vstup z kola `t`. K_t je konštanta pre kolo `t` a $\boxplus$ znamená sčítanie spolu s modulom 2^{32}.

[2] Zdroj: https://en.wikipedia.org/wiki/MD5



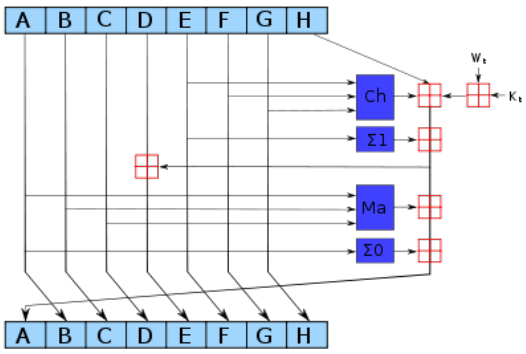
Obr. 4.7: Jedno kolo algoritmu SHA-1 [3]

4.2.3 SHA-256

SHA-256 [4] je jedna z hašovacích funkcii zo skupiny algoritmov SHA-2.

Algoritmus vykonáva 64 kôl nad 512 bitovými blokmi podobne ako u SHA-1. Tak isto robí zarovnanie vstupu. Líši sa veľkosťou vnútorného stavu, ktorý je 256 bitový. Obsahuje 8 vnútorných blokov (A-H) po 32 bitov ktoré sú nazačiatku inicializované určenými konštantami.

Na obrázku 4.2.3 je znázornene jedno kolo zo skupiny funkcii SHA-2. A-H sú vnútorne stavy. `Ch` a `Ma` sú kompresné funkcie obsahujúce operácie XOR a AND. $\sum 0$ a $\sum 1$ sú funkcie, ktoré vykonávajú operácie XOR a posun bitov vpravo. W_t je rozšírený vstup z kola `t`. K_t je konštanta pre kolo `t` a $\boxplus$ znamená sčítanie spolu s modulom 2^{32}.



Obr. 4.8: Jedno kolo algoritmu zo skupiny SHA-2 [4]

[3] Zdroj: https://en.wikipedia.org/wiki/SHA-1
[4] Zdroj: https://en.wikipedia.org/wiki/SHA-2

Kapitola 5

Rozbor formátov

V tejto kapitole rozoberieme každý formát súboru, akú signatúru obsahujú pre detekciu daného formátu. Popíšeme aké údaje sa kde nachádzaju a ktoré z nich sú potrebné pre obnovu hesla.

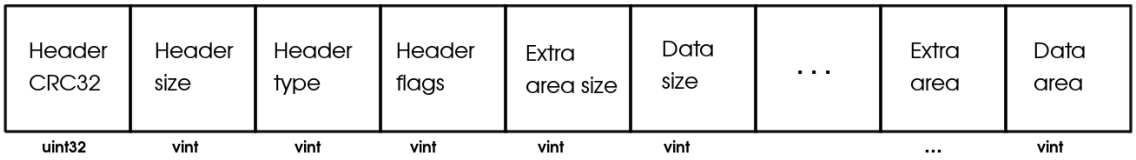
5.1 RAR

RAR je formát súboru určený pre kompresiu a archiváciu súborov. Ďalej umožňuje aj opravu chýb, rozdelenie súboru na viac súborov, ale najmä zašifrovanie archívu. V archíve môžme zašifrovať samotné dáta súborov, ale aj hlavičku súboru, takže bez znalosti hesla nevieme zistiť ani len aké súbory sa v archíve nachádzajú. V minulosti sa na šifrovanie používalo *AES-128* (viz sekciu 4.1.1). Aktuálne od najnovšej revízie 5.0 sa používa *AES-256*. Revíziu súboru môžme určiť na základe špecifickej signatúry na začiatku súboru.

5.1.1 RAR 5.0

Revízia 5.0 [10] obsahuje na začiatku súboru signatúru `0x52 0x61 0x72 0x21 0x1A 0x07 0x01 0x00`. Štruktúra súboru sa skladá z archivačných blokov. Každy blok obsahuje hlavičku. Hlavička sa skladá z následujúcich položiek:

- Header CRC32 - kontrolný súčet od `Header size` až po nepovinnú `Extra area`
- Header size - obsahuje veľkosť celej hlavičky od `Header type` po `Extra area`. Maximálna veľkosť tohto poľa je 3 bajty.
- Header type - značí o aký typ hlavičky sa jedná. Môže nadobúdať 5 hodnôt:
  - Main archive header - obsahuje `Archive flags` ktorý môže obsahovať príznak `0x8`, ktorý značí ze archív je zamknutý.
  - File header - obsahuje informácie o súbore: veľkosť, meno, samotné dáta...
  - Service header - používa štruktúru ako `File header` na uchovanie doplnkových informácii.
  - End of archive header - označenie konca archívu.
  - Archive encryption header

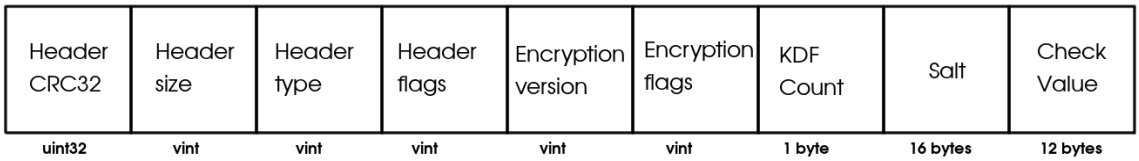


Obr. 5.1: Štruktúra RAR formátu

Typ `vint` môže obsahovať jeden alebo viac bajtov, kde najmenej významných 7 bitov každého bajtu obsahuje číselne dáta a najvýznamnejší bit obsahuje príznak či bude obsahovať ďalšie číselne dáta.

Archive encryption header sa vyskytuje len v archívoch so zašifrovanou hlavičkou. Ako každá hlavička obsahuje kontrolný súčet, veľkosť, príznaky a typ hlavičky ktorý je v tomto prípade `0x4`. Ďalej obsahuje:

- Encryption version - verzia algoritmu pre šifrovanie, aktuálne podporovaná je len 0, ktorá značí *AES-256*.
- Encryption flags - len príznak `0x1` ktorý označuje použitie hesla.
- KDF count - binárny logaritmus počtu iterácii pre *PBKDF2* funkciu.
- Salt - 16 bajtová soľ použitá pre každú zašifrovanú hlavičku.
- Check value - 12 bajtová hodnota použitá pre overenie hesla. Prvých 8 bajtov je vypočítaných ďalšími iteráciami *PBKDF2*. Posledné 4 bajty sú ďalší kontrolný súčet.



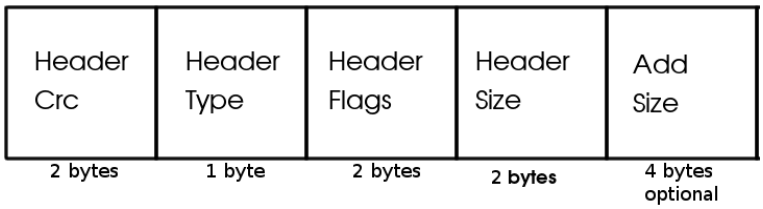
Obr. 5.2: Štruktúra šifrovacej hlavičky formátu RAR

Pre každý zašifrovaný súbor v archíve sa vyskytuje záznam ktorý je podobný archivačnej šifrovacej hlavičke, ale obsahuje aj `IV` - ktorý slúži na inicializáciu *AES-256*.

Obr. 5.3: Štruktúra záznamu pre zašifrovaný súbor vo formáte RAR 5.0

5.1.2 RAR 3.0

Revízia 3 obsahuje na začiatku súboru signatúru `0x52 0x61 0x72 0x21 0x1A 0x07 0x00`. Skladá sa z blokov rôznych typov, podobných ako u RAR 5.0. Každý blok obsahuje hlavičku, ktorá špecifikuje o akú hlavičku sa jedná a jej veľkosť.



Obr. 5.4: Štruktúra hlavičky vo formáte RAR 3.0

V hlavnej hlavičke archívu (*type == 0x73*) sa v príznakoch nachádza príznak (*0x04*) ktorý značí zašifrovanie aj hlavičky, tým pádom nepoznáme ani mená súborov nachádzajúce sa v archíve.

Narozdiel od novšej revízie sa nedá rozlíšiť či je zadané heslo nesprávne, alebo je RAR súbor poškodený. Zo zadaného hesla a náhodne vygenerovanej soli uložená v archíve, sa vygeneruje pomocou *SHA-1*4.2.2 inicializačný vektor a kľúč pre *AES-128*.

Pre overenie hesla je nutné dekomprimovať súbor, vypočítať jeho *CRC*[1] a porovnať ho s *CRC* v RAR súbore. Ak sa nezhodujú tak heslo není validné. Môže to ale značiť poškodený archív.

5.2 GZIP

GZIP [2] je algoritmus pre bezstratovú kompresiu dát. Je založený na algoritme *deflate* ktorý používa *LZ77* algoritmus a Huffmanove kódovanie. Tento algoritmus není pokrytý žiadnym patentom a preto sa rozšíril, okrem GZIPu ho používa aj PNG a ZIP formát. Prípona súboru je `.gz`. Samotný formát neposkytuje šifrovanie a archiváciu. Na archiváciu sa zvyčajne používa súborový formát *tar*.

GZIP formát pozostáva zo série blokov idúcich za sebou. Každy blok obsahuje bajty `ID1` a `ID2`, ktoré identifikujú formát. Za ním nasleduje `CM` - *Compression Method* ktorá označuje metódu použitú pre kompresiu. Momentálne je podporovaná len *deflate* ktorá ma hodnotu `0x8`, 0 až 7 sú rezervované. Spolu s ID1 a ID1 teda môžme určiť:
Signatúru: `0x1F 0x8B 0x08`
Ďalej obsahuje bajt `FLG` ktoré obsahuje príznaky a bity od 0 po 4, 5 až 7 sú rezervované a musia byť nastavené na 0.

- FTEXT - označuje ze sa jedná pravdepodobne o ACII text.
- FHCRC - ak je nastavený, tak pred komprimovanými dátami sa nachádza *CRC16* dva LSB z *CRC32* pre všetky bajty gzip hlavičky po *CRC16*.
- FEXTRA - ak je nastavený, tak hlavička obsahuje extra polia.
- FNAME - označuje, že je uložene originálne meno súboru.
- FCOMMENT - komentár ktorý není interpretovaný.

Po `FLG` nasleduje 4 bajtový `MTIME` - *Modification time*, v unixovom formáte udáva kedy naposledy bol komprimovaný súbor zmenený.
Ďalej `XFL` - *Extra flags* - je špecifický pre metódu kompresie. Momentálne môže obsahovať hodnoty len pre *deflate*, 2 označuje ze bola použitá maximálna kompresia a teda najpomalší

[1] Cyclic redundancy check

algoritmus, 4 označuje ze bol použitý najrýchlejší algoritmus.
Nasleduje `OS` - *Operating system* - obsahuje identifikačné číslo operačného systému na ktorom bola kompresia vykonaná. `XLEN` - *Extra length* - udáva dĺžku voliteľného extra pola. Za tým nasledujú skomprimované bloky súboru. Za nimi je `CRC32` - Kontrolná hodnota neskomprimovaného súboru. A nakoniec `ISIZE` - veľkosť originálneho súboru s modulom 2^{32}.

5.3 BZIP2

BZIP2 [12] je program pre bezstratovú kompresiu dát. Komprimuje len súbor a teda neslúži na archiváciu. Používa *Burrows-Wheeler* algoritmus, ktorý je efektívnejší ako *deflate* použitý v GZIP. Na druhú stranu je ale omnoho pomalší. Samotný formát neposkytuje šifrovanie. Formát obsahuje dáta, ktoré slúžia na kompresiu, tieto dáta sú pre naše účely bezvýznamne a nebudeme ich ďalej skúmať.
Signatúra: `0x42 0x5A 0x68`

Kapitola 6

OpenSSL

OpenSSL [1] je open-source softwarová knižnica používaná pre bezpečnú komunikáciu. OpenSSL projekt bol založený v roku 1998 za účelom poskytnutia šifrovacích nástrojov. Je založená na SSLeay, ktorého ale vývoj skončil koncom roka 1998. Projekt riadi celosvetová komunita dobrovoľníkov ktorí používajú internet na komunikáciu, plánovanie a vývoj OpenSSL a jej dokumentácie.

OpenSSL implementuje SSL (*Secure Sockets Layer*) a TLS (*Transport Layer Security*) a taktiež širokú škálu šifrovacích algoritmov: Blowfish, Cast, DES, 3DES, RC2, RC4, RC5, ... Ďalej podporuje aj kryptografické hašovanie : MD5, MD4, SHA-1, SHA-2, ... a aj asymetrickú kryptografiu: RSA, DSA, ...

OpenSSL šifruje celý súbor, nie len samotné dáta, tým pádom nám stačí dešifrovať len prvý blok a porovnať signatúru súboru.

[1] https://www.openssl.org/docs/

Kapitola 7

Návrh modulov

V tejto kapitole budeme popisovať ako modul deteguje zadaný archív a ako ho následne spracuje a uloží informácie nutné pre overenie hesla. Nakoniec ako modul overí, či je získane heslo z generátora správne s použitím informácii získaných pri spracovaní súboru.

7.1 RAR 5.0

Formát RAR 5.0 používa príponu .rar a signatúru `0x52 0x61 0x72 0x21 0x1A 0x07 0x01 0x00`. Jedná sa o binárny formát, takže jeho čítanie je pomerne jednoduché, ale formát obsahuje veľa voliteľných hlavičiek a položiek ktoré komplikujú čítanie. Ak je archív šifrovaný tak po signatúre sa nachádza *Archive encryption header* (viz 5.1.1) ktorý obsahuje väčšinu položiek typu `vint` ktorý môže byť v rozsahu od 8 po 64 bitov, aj keď väčšinou obsahuje len 8 bitov z toho 7 dátových. Zo záznamov `size`, `type` a `flags` zistíme informácie vďaka ktorým sa budeme vedieť pohybovať v súbore. Záznamy `encryption version, encryption flags, KDF count, salt` a `check value` si uložíme pre účely obnovy hesla.

Nasleduje Main archive header z ktorý len preskočíme, položky z neho použijeme len na korektný pohyb v súbore. Hneď za hlavnou hlavičkou archívu sa môže vyskytovať *Archive comment header*, ktorý pre naše účely nepotrebujeme a len ju preskočíme.

Pre každý súbor v archíve sa vyskytuje hlavička, ktorá obsahuje informácie o danom súbore. Za hlavičkou sa môžu vyskytovať tzv. *service headers* ktoré obsahuje dodatočne informácie. Nás bude zaujímať *file encryption record*, ktorý obsahuje takmer totožne dáta ako *archive encryption header*, len naviac obsahuje inicializačný vektor `IV` slúžiaci pre AES-256.

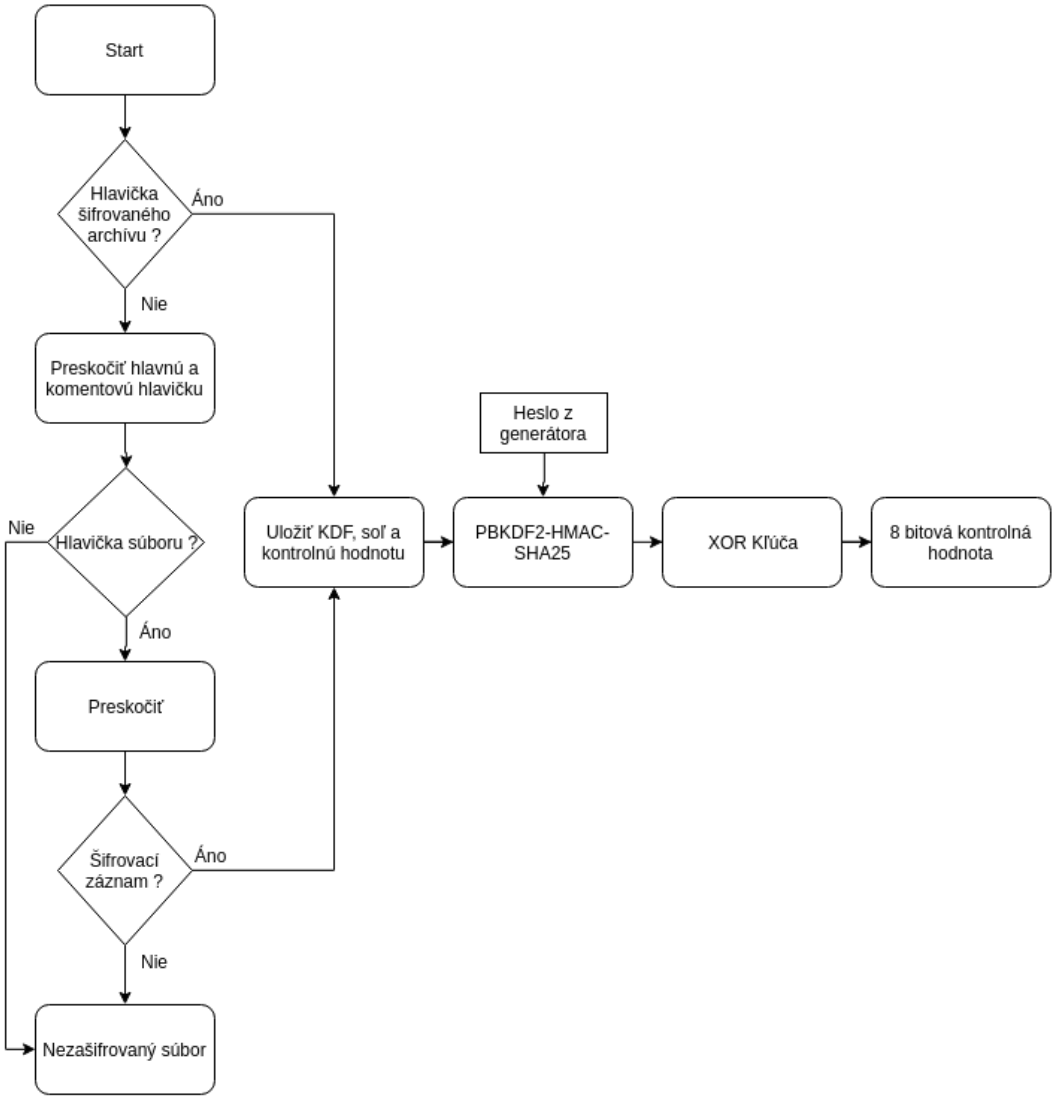
Pred koncom súboru sa môže nachádzať *recovery record*, ktorý je pre obnovu hesla nepodstatný. Na samotnom konci sa nachádza už len *end of archive header*, ktorý označuje koniec archívu.

Po získaní potrebných informácii môžme čítanie archívu skončiť a začať obnovu hesla. Pomocou získanej soli (angl. *salt*) a hesla z generátoru hesiel sa vykoná $2^{KDF} + 32$ iterácii PBKDF2-SHA256[1] a získame 32 bitový kľúč. Následne jednotlivé bity XOR-ujeme a získame 8 bitovú hodnotu, ktorú porovnáme s `CheckValue`, získanú zo súboru. Ak sa hodnoty zhodujú tak kontrolované heslo je správne.

[1] Password-Based Key Derivation Function 2 - Secure Hash Algorithm

Pseudokód 7.1 Kontrola správnosti hesla vo formáte RAR 5.0

```
1: bool checkRARPass (string salt, string password) {
2:         char key[32];
3:         int iterations = 1 << KDF + 32;
4:         for (int i = 0; i < iterations; i++)
5:                 PBKDF2_SHA256(key,salt,password);
6:         char check[8] = {0};
7:         for (int i = 0; i < 32; i++)
8:                 check[i % 8] ^= Key[i];
9:         return check == CheckValue&0xFF;
10: }
```



Obr. 7.1: Kroky pre získanie kontrolnej hodnoty pre overenie hesla.

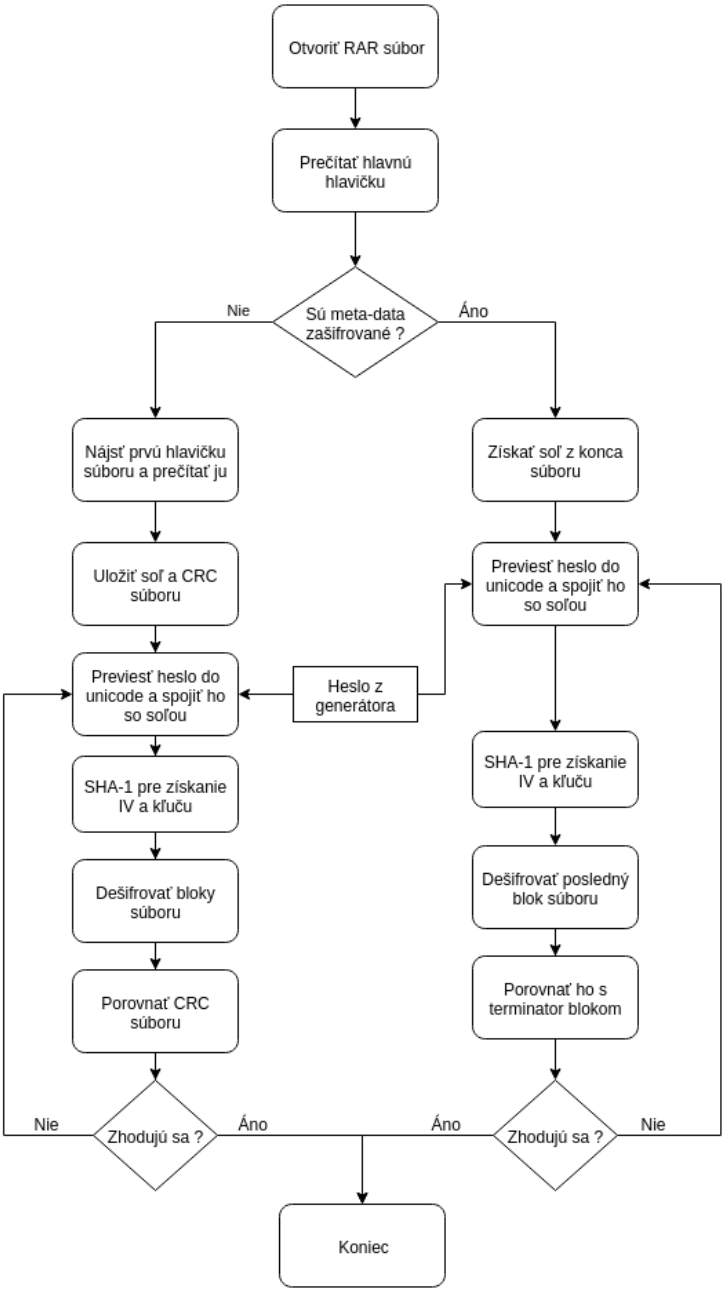
7.2 RAR 3.0

V RAR 3.0 sa nazačiatku súboru vyskytuje signatúra `0x52 0x61 0x72 0x21 0x1A 0x07 0x00`.

Ak archív nemá zašifrovanú hlavičku, prejdeme na prvý záznam o súbore v archíve. Prečítame z neho príznaky, príznak `0x04` značí zašifrovaný súbor. Ak sa nenachádza, tak archív není zaheslovaný. V opačnom prípade si uchováme 4 bajtový `FILE_CRC` a 8 bajtový `SALT`. Pre šifrovanie sa používa *AES-128*, kľúč a inicializačný vektor pre tento algoritmus sa vygeneruje zo zadaného hesla pomocou *SHA-1* spolu zo získanej soli, ktorý sa prevedie 262144 krát [3].

Následne je nutné dešifrovať celý súbor a porovnať CRC súboru s predtým získaného CRC z archívu. Ak sa zhodujú, heslo je správne. Ak nie, tak heslo je nesprávne alebo zadaný archív je poškodený. Keďže tieto 2 informácie nemôžeme rozlíšiť, musíme skúšať ďalšie heslá.

Pri zašifrovaní hlavičky, sa zašifrujú všetky meta dáta, vrátane tzv. *terminator*. Je to blok ktorý značí koniec RAR archívu. Jeho obsah je vždy rovnaký, hexadecimálne `0xC4 0x3D 0x7B 0x00 0x40 0x07 0x00`. Takže nám stačí dešifrovať posledný blok a porovnať koniec bloku s *terminator*. Nemúsime dešifrovať celý súbor a počítať CRC, takže sa nám proces obnovy hesla zrychlí.



Obr. 7.2: Overenie validity hesla v RAR 3.0

7.3 OpenSSL

Formáty BZIP2 a GZIP nemajú možnosť šifrovania na jeho šifrovanie sa použije OpenSSL ktoré zašifrujú celý súbor vrátane signatúry. K dispozícii je veľa šifier, ktoré je možné použiť. My sa budeme zaoberať s *AES-256-CBC*.

OpenSSL šifruje celý súbor a po samotnom zašifrovaní nemáme dostupné žiadne informácie o použitom algoritme. Algoritmus šifrovania pred samotnou obnovou hesla je nutné

špecifikovať vrátené formátu archívu. Prvý blok sa pomocou zadaného hesla dešifruje a porovná sa signatúra súboru, ak sa bude zhodovať tak heslo je správne.

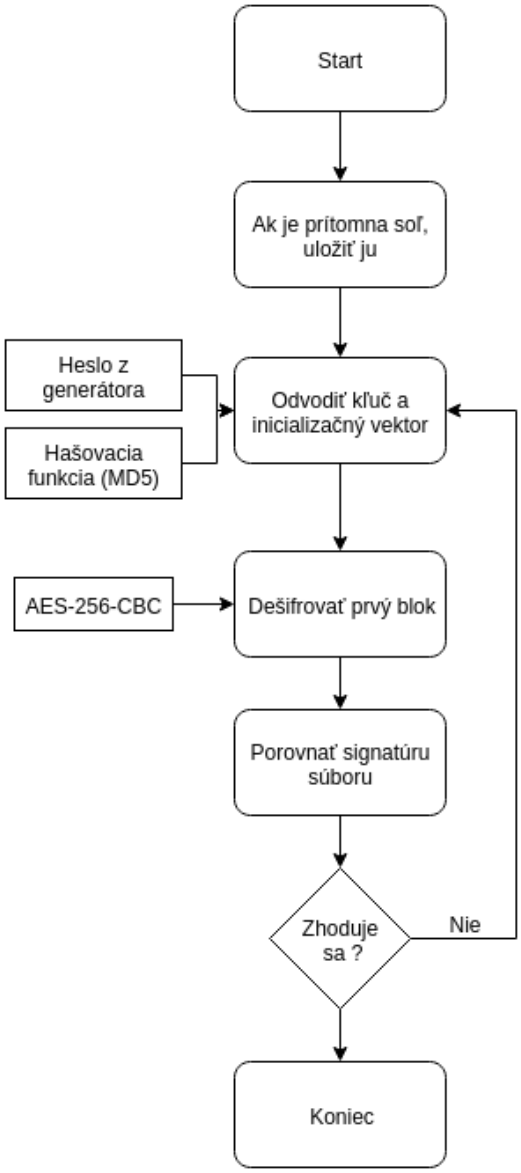
Jedinú informáciu, ktorú zo súboru môžme dostať je soľ. Ak bola použitá soľ, tak prvých 8 bajtov súboru je text `Salted__` hexadecimálne `0x53 0x61 0x6C 0x74 0x65 0x64 0x5F 0x5F` a za ním nasleduje samotná 8 bajtová soľ. Následne sa nachádzaju už len samotné zašifrované dáta.

OpenSSL používa vlastný algoritmus na odvodenie inicializačného vektora a kľúča. Inicializačný vektor a kľúč sa generujú zo zadaného hesla. Keď sa explicitne nedefinuje hašovacia funkcia, tak sa použije MD5. Prvých 16 bitov kľúča sa získa ako výsledok hašovacej funkcie, kde sa ako parameter predá heslo a soľ. Na týchto 16 bitov sa spolu s heslom a soľou použije znova hašovacia funkcia a získame posledných 16 bitov kľúča. Inicializačný vektor získame z posledných 16 bitov kľúča, hesla a soli.

Pseudokód 7.2 Odvodenia hesla a inicializačného vektoru z hesla

```
 1: void derivateKeyAndIV(string password, string salt,
 2:  char[] key, char[] IV){
 3:     char result[16];
 4:     char firstHalf[16];
 5:     result = hash_function(password + salt);
 6:     firstHalf = result;
 7:     result = hash_function(result + password + salt);
 8:     key = firstHalf + result;
 9:     iv = hash_function(result + password + salt);
10: }
```

V pseudokóde 7.2 vidíme ako získať výstup `key` a `IV` pre šifrovací algoritmus zo vstupu `password` a `salt`. Po získaní kľúča a inicializačného vektora sa použije samotný šifrovací algoritmus ktorým dešifrujeme prvý blok dát a porovnáme začiatok dát so signatúrou, ak sa zhoduje tak prehlásime dané heslo za správnym.



Obr. 7.3: Kroky pre overenie hesla súboru zašifrovaného pomocou OpenSSL

Kapitola 8

Implementácia

V tejto kapitole rozoberieme implementáciu jednotlivých modulov. V jazyku C++ je implementované prečítanie dát z archívu a obnova hesla pre CPU. V jazyku OpenCL C je napísaná implementácia kernelu pre GPU. Implementované boli následovne moduly:

1. RAR
2. OpenSSL

8.1 RAR

Pri čítaní archívu sa volá metóda `readOneFile` ktorá prečíta prvých 6 bajtov a porovná sa či sa jedná o RAR archív. Následne sa prečíta ďalší bajt, pokiaľ je to `0` jedná sa o RAR 3 archív, ak `1` a za ním nasleduje `0` tak ide o RAR 5 archív. Tieto archívy majú rôznu štruktúru, pri RAR 3 sa volá metóda `readV3` a pri RAR 5 metóda `readV5` ktoré vytiahnu z archívu potrebné dáta pre obnovu hesla.

8.1.1 RAR 5

O načítanie dát z tejto verzie archívu sa stará funkcia `readV5`. V tomto archíve sa vyskytuje typ `vint` ktorý má premenlivú dĺžku. Pre tento účel bola implementovaná trieda a metóda `VInt::read` ktorá číta jednotlivé bajty z `std::ifstream` pokiaľ je nutné a následne uloží do typu `uint64_t`.

Táto funkcia hľadá šifrovaciu hlavičku, ak ju nájde tak z nej vytiahne a uloží soľ, overovaciu hodnotu a počet kôl, ktoré sa majú vo funkcii PBKDF2. Ak sa nenájde, tak sa hľadá hlavičkový súbor so šifrovaným záznamom z ktorej sa vytiahnu rovnaké údaje ako zo šifrovacej hlavičky. Nachádza sa tam aj inicializačný vektor, ktorý ale pre overenie hesla nie je potrebný.

Po získaní potrebných údajov sa vykoná potrebné množstvo kôl funkcie PBKDF2-HMAC-SHA256 [6, 9, 4]. Po tomto množstve kôl dostaneme kľúč pre AES-256, ktorý nepotrebujeme pre účely overenie hesla. Po ďalších 32 kolách získame 32 bajtový `pswcheck`. Následne sa tieto bajty XOR-ujú, 0. bajt s 8, 16 a 24 atď. Z toho vznikne 8 bajtová kontrolná hodnota, ktorú porovnáme s hodnotou získanú z archívu.

Pre GPU variantu bol napísaný kernel `rar5_kernel` ktorý vykonáva to isté ako CPU varianta.

8.1.2 RAR 3

O načítanie dát z tejto verzie archívu sa stará funkcia `readV3`, ktorá nájde hlavnú hlavičku a zistí z nej či boli zašifrované aj mená súborov, čiže meta-dáta.

Ak áno tak z konca súboru sa uloží posledných 24 bajtov, kde 8 bajtov je soľ a 16 bajtov sú dáta koncovej hlavičky a prechádza sa na samotnú obnovu hesla.

Ak neboli zašifrované meta-dáta, tak sa prechádza celý archív a hľadá sa hlavička súboru ktorá je zašifrovaná. Skúma sa každý súbor v archíve a potrebné dáta vyberie z toho ktorú ma najmenšiu veľkosť. Z neho si uloží údaje o veľkosti súboru, použitej kompresii, kontrolný súčet súboru a soľ

Po získaní potrebných dát nasleduje samotná obnova hesla. Pre overenie hesla je potrebné heslo z generátora previesť do UTF-16 kódovania. Následne funkcia `genAesValues` vygeneruje kľúč a inicializačný vektor pre AES-128. Heslo v UTF-16 kódovaní sa spojí so soľou a je vykonané 262144 kôl hašovacej funkcie SHA-1 (viz 4.2.2). Kde každé 16384. kolo je použité pre zostrojenie inicializačného vektoru. A samotný výsledok je použitý ako kľúč.

Ak boli zašifrované meta-dáta, tak pomocou algoritmu AES-128 dešifrujeme 16 bajtový blok dát z konca archívu a porovnáme s bajtami koncovej hlavičky, ak sa zhodujú tak heslo z generátora bolo správne.

Ak neboli zašifrované meta-dáta, tak pred samotným dešifrovaním je nutné najprv spraviť dekompresiu dát, ak bola použitá kompresia, táto možnosť nebola implementovaná. Následne sa dešifrujú samotné dáta a vypočíta sa z nich kontrolný súčet, ktorý sa porovná s kontrolným súčtom získaného z archívu.

Boli implementované 2 podobné kernely v OpenCL C. `rar3_header_kernel` ktorý sa stará o obnovu hesla pre variantu so zašifrovanými meta-dátami. A `rar3_kernel` ktorý potrebuje viac pamäti kvôli dešifrovaniu celého súboru a počíta kontrolný súčet ktorý porovnáva.

8.2 OpenSSL

Pre OpenSSL modul bolo nutné upraviť jadro Wrathionu. Boli pridané 4 parametre:

- –encryptions - Vypíše podporované šifry (AES)
- –formats - Vypíše podporované formáty (GZIP, BZIP)
- –encryption=cipher - Zvolí šifru ktorá sa použije
- –format=format - Zvolí formát zadaného súboru

Do triedy `FormatPool` bola pridaná metóda `getOpenSSLFormat` ktorá sa pokúsi nájsť OpenSSL formát, inak vracia `nullptr`. Takto sa kontroluje či je dostupný OpenSSL modul, ak by užívateľ chcel použiť parameter týkajúci sa tohto modulu.

V triede `FileFormat` boli pridané metódy `getCiphers` a `getFormats`, ktoré vracajú `nullptr`, tieto metódy potom reimplementuje OpenSSL modul, ktorý vráti mapu s dostupnými šiframi a formátmi.

V triede `OpenSSLFormat` sú v premenne `ciphers` definované podporované šifry. V premenne `formats` typu `std::map<std::string, std::string>` sú definované podporované formáty, kde na meno formátu je namapovaná jeho signatúra, týmto spôsobom je ľahké doplniť ďalšie formáty.

V metóde `init` sa potom inicializujú potrebné údaje, ako vstup je jeden reťazec, ktorý obsahuje meno súboru, šifru a formát súboru oddelené bodkočiarkou. Následne sa otvorí súbor a zavolá sa metóda `readOneFile` ktorá prečíta prvých 16 bajtov. Ak sa na prvých 8 nachádza reťazec `Salted__` tak ďalších 8 sa uloží ako soľ. Ak nie, tak metóda končí a prečítané dáta uloží. Ak sa v súbore vyskytovala soľ, tak sa prečíta ďalších 16 bajtov, ktoré sa uložia a budú použité pri dešifrovaní.

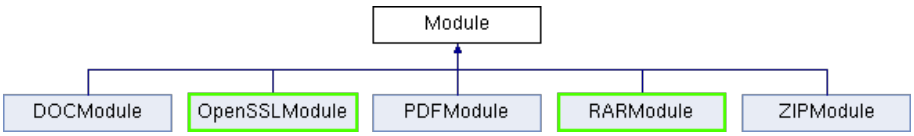
Bola implementovaná šifra AES v 128, 192 a 256 bitovým kľúčom v CBC režime. Pri overení hesla sa soľ pripojí na koniec hesla z generátora. V metóde `deriveKeyAndIV` sa z hesla so soľou odvodí kľúč a inicializačný vektor pre AES pomocou hašovacej funkcie MD5 (viz 4.2.1). Prvých 16 bajtov dostaneme jedným kolom hašovacej funkcie. Ďalších 16 bajtov sa získa znova použitím hašovacej funkcie a ako vstup bude výsledok predošlého výpočtu, ku ktorému sa pridá heslo so soľou. Ak potrebujeme ďalšie bajty (pri AES-192 a 256), tak sa tento krok vykoná ešte raz. Tým dostaneme 48 bajtov. Prvých 16 slúži ako inicializačný vektor a zvyšné ako kľuč, pre AES-128 je to 16 bajtov, AES-192 24 bajtov a pre AES-256 celých 32 bajtov.

Keď máme kľúč a inicializačný vektor, tak dešifrujeme prvý blok súboru získaného zo súboru. Potom sa už len porovná dešifrovaný blok so signatúrou.

Signatúra formátov GZIP a BZIP2 je len 3 bajtová, tým pádom by sme veľmi ľahko našli zhodu aj pri zle zadanom hesle. Aby sme sa vyhli falošne pozitívnom hesle, je nutné vykonať kontrolu ďalších bajtov. Pri formáte BZIP sa od 5. bajtu vyskytuje ďalších 6 fixných bajtov `0x31 0x41 0x59 0x26 0x53 0x59`, toto nám už stačí pre potvrdenie hesla. Pri formáte GZIP sa ale nič také nevyskytuje, preto ako ďalšia kontrola bola použitá overovanie rozsahu jednotlivých bajtov.

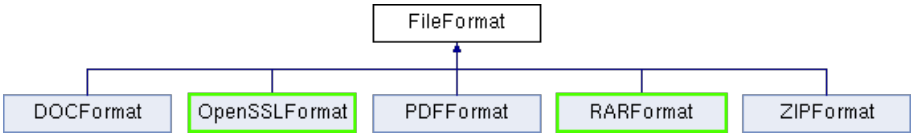
8.3 Integrácia

Najprv bolo nutné vytvoriť `RARModule` a `OpenSSLModule` ktoré dedia z `Module` a obsahujú meno autora, popis modulu a `FileFormat`.



Obr. 8.1: Integrácia nových modulov

Pre každý modul bol implementovaný `FileFormat`, ktorý sa stará o prečítanie súboru a vytiahnutie potrebných dát. Ide o `RARFormat` a `OpenSSLFormat` ktoré ešte vracajú továrne pre obnovu hesla, ako CPU variantu tak aj GPU.

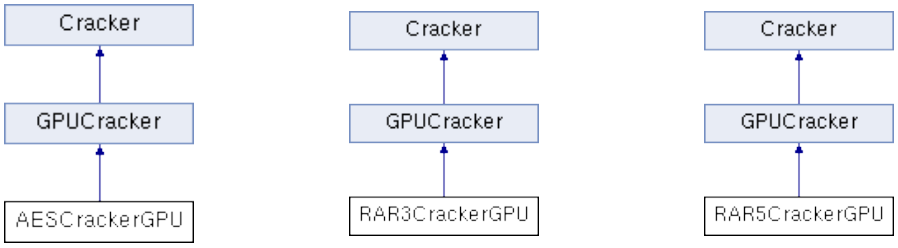


Obr. 8.2: Integrácia nových formátov

Napokon boli pridané samotné továrne, ktoré sa starajú o obnovu hesla a samotné crackery. Pre CPU variantu sú to `AESCrackerCPU`, `RAR3CrackerCPU` a `RAR5CrackerCPU` a implementujú metódu `checkPassword` z triedy `Cracker` ktorá dostane ako vstup heslo a overí či je

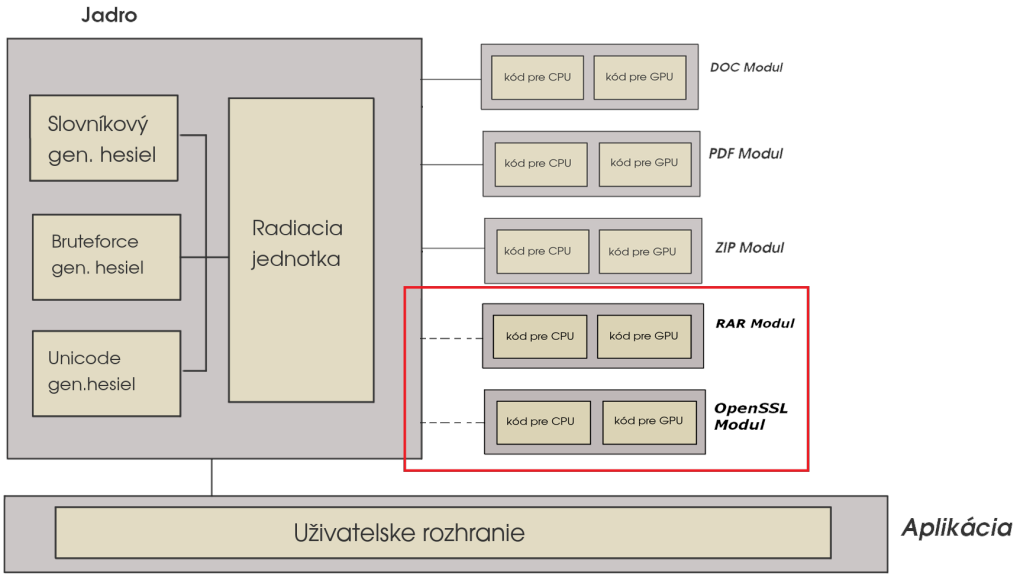
správne.

Pre GPU variantu boli implementované `AESCrackerGPU`, `RAR3CrackerGPU` a `RAR5CrackerGPU`. Tieto triedy implementujú metódu `init` ktorá sa stará o predanie dát z CPU do GPU kernelu. V konštruktore sa definuje premenná `kernelFile` ktorá udáva v ktorom súbore sa nachádza kernel a premennú `kernelName` v ktorej je názov samotného kernelu.



Obr. 8.3: Diagram tried

Neimplementované moduly boli začlenené do architektúry nástroja Wrathion, ktoré rozšírili jeho funkcionalitu. K pôvodnému ZIP, PDF a DOC modulu boli pridané RAR a OpenSSL moduly.



Obr. 8.4: Nové moduly v nástroji Wrathion

Kapitola 9

Experimenty a testovanie

V rámci experimentov sme testovali výkonnosť implementovaných modulov. Ako generátor hesiel bol použitý útok hrubou silou s použitím 26 znakov od `'a'` po `'z'`. Experimenty boli vykonávane na rôznych dĺžkach hesla, pričom dané heslo bolo ako posledné v danej sekvencii, napríklad pre 3 znakový test to bolo heslo `zzz`. Aby sme našli `k`-znakové heslo je nutné vygenerovať všetky kombinácie podľa rovnice 9.1 a overiť ich.

$$\sum_{n=1}^{k} 26^n \tag{9.1}$$

Budeme merať rýchlosť jednotlivých modulov pri rôznych variantách. Rýchlosť predstavuje koľko hesiel je nástroj schopný vygenerovať a následne ich overiť za jednu sekundu.

9.1 Testovacia zostava

CPU: Intel Core i-7-5930K
GPU: 4x GIGABYTE R9 Fury X 4GB HBM
RAM: 32GB DDR4
OS: Windows 8.1 Pro N

9.2 OpenSSL

V rámci OpenSSL modulu sme experimentovali s kryptografickým algoritmom AES s dĺžkou kľúča 128, 192 a 256 bitov.

9.2.1 CPU Škálovateľnosť

Testy sa vykonávali na 1, 2 a 4 vláknach pri ktorých sa zaznamenala celková rýchlosť.

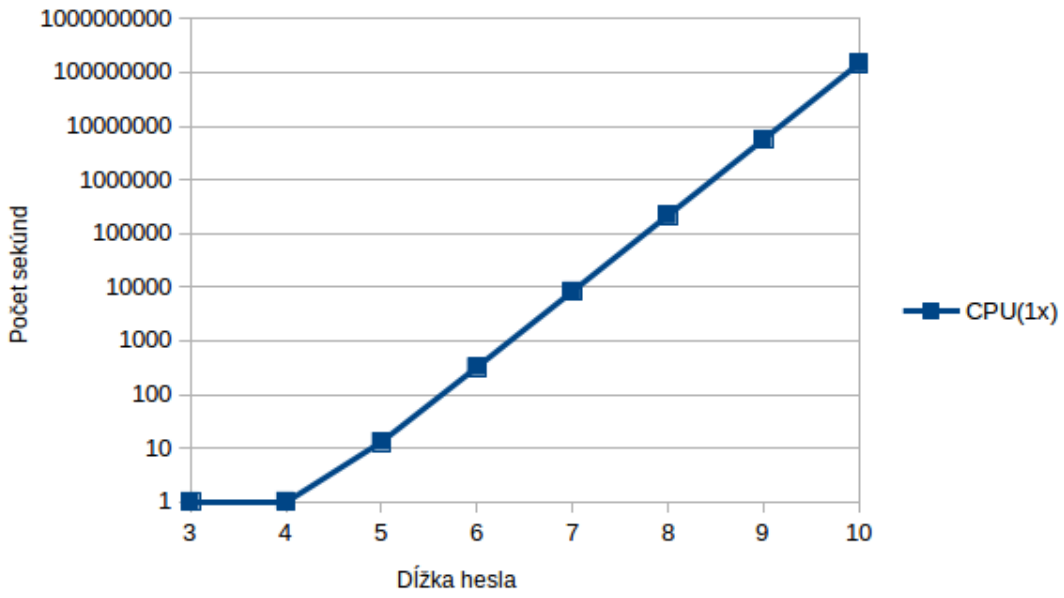
	1 Vlákno	**2 Vlákna**	**4 Vlákna**
Rýchlosť	1003983	1472268	1123470

Tabuľka 9.1: Rýchlosť overovania hesla pri AES-128 CPU varianta

Mohli by sme sa domnievať, že pri zdvojnásobení počtu vlákien sa nám zdvojnásobí aj rýchlosť. Ale ako môžeme vidieť v tabuľke 9.1 tak sa to nestane, pri dvoch vláknach sa

nám rýchlosť zvýši len o 46%. Pri 4 vláknach môžeme pozorovať nižšiu rýchlosť ako pri 2 vláknach. Je to zapríčinené tým, že rýchlosť tohto modulu vysoká a réžia synchronizácie vlákien nestíha v rozumnom čase prideľovať každému vláknu heslá.

Pri zvyšovaní dĺžky hesla môžeme očakávať, že čas pre nájdenie o 1 znak dlhšieho hesla bude trvať 26 násobne dlhšie. Ako môžeme vidieť na grafe 9.1 tak to platí. Nájdenie 6 miestneho hesla trvá 322 sekúnd a pri 6 miestnom to je 8319 sekúnd, čo je 26 násobne dlhšie.



Obr. 9.1: Rýchlosť nájdenia hesla pri rôznych dĺžkach hesla pri AES-128 CPU 1 vlákno

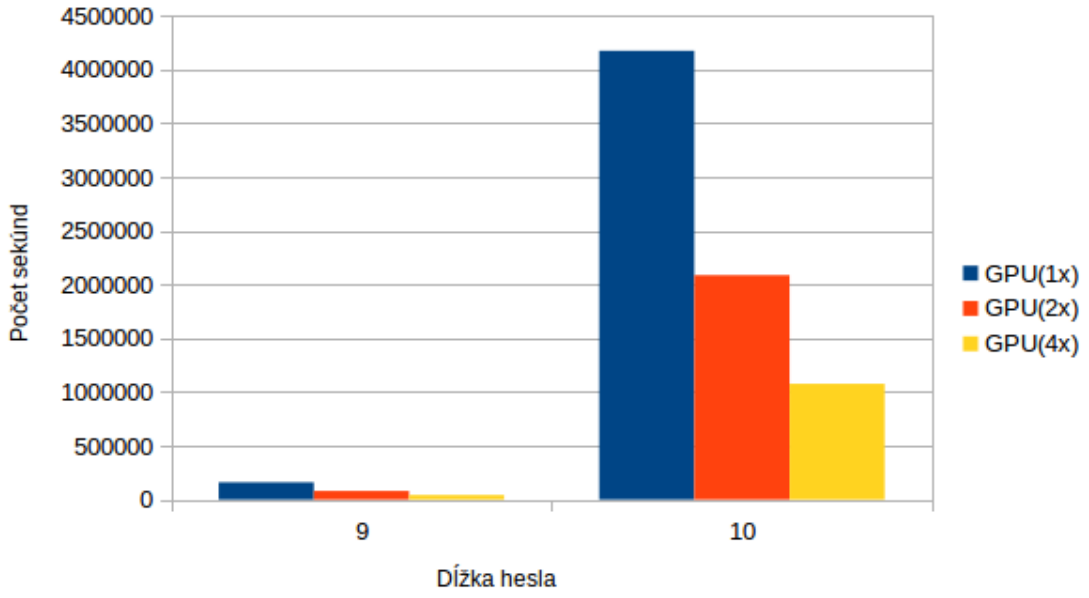
9.2.2 GPU Škálovateľnosť

Testy sa vykonávali na 1, 2 a 4 rovnakých grafických kartách. Narozdiel od CPU varianty, pri GPU môžeme pozorovať zdvojnásobení výkon pri pridaní ďalšej grafickej karty. Tak isto, keď zvýšime počet GPU z 2 na 4, tak sa rýchlosť zdvojnásobí. Keďže sa rýchlosť

	1x GPU	**2x GPU**	**4x GPU**
Rýchlosť	35174878	70351210	136368360

Tabuľka 9.2: Rýchlosť overovania hesla pri AES-128 GPU varianta

zdvojnásobuje, tak aj čas potrebný na nájdenie hesla je dvojnásobne menší ako môžeme vidieť na grafe 9.2.



Obr. 9.2: Porovnanie času potrebného pre nájdenie hesla pre rôzny počet GPU

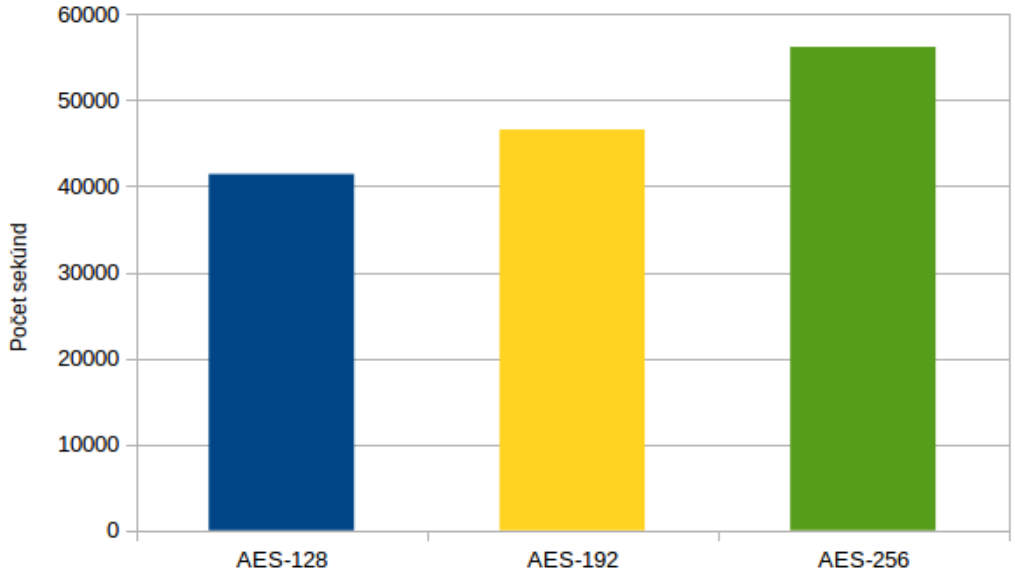
9.2.3 Porovnanie dĺžky kľúča

Pri dlhšom kľúče treba vykonávať viac kôl algoritmu a taktiež pri šírke kľúča 192 a 256 bitov je nutné vykonať o jedno kolo hašovacej funkcie naviac, takže môžeme predpokladať, že rýchlosť bude menšia ako pri 128 bitov.

	AES-128	**AES-192**	**AES-256**
Rýchlosť	136368360	121212151	100502946

Tabuľka 9.3: Rýchlosť overovania hesla pri 4x GPU

V tabuľke 9.3 vidíme, že rýchlosť pri AES-192 sa oproti AES-128 spomalila o 12.5%. AES-256 je oproti AES-192 o 12% pomalší a oproti AES-256 o 35.6% pomalší. Na grafe 9.3 môžeme vidieť potrebný čas na nájdenie 9 miestneho hesla pri rôznej dĺžke kľúča.



Obr. 9.3: Porovnanie času potrebného pre nájdenie 9 miestneho hesla pri 4x GPU

9.2.4 Zhrnutie

Pri testoch na zistil problém pri formáte GZIP, dešifruje sa len prvý blok a porovnávajú sa 3 bajty signatúry a následne sa overuje rozmedzie ďalších bajtov. Aj pri tomto ďalšom ošetrení nastalo, že pri 10 miestnom hesle sa našlo falošne pozitívne heslo, nemáme už dostupné ďalšie hodnoty na porovnanie, tak tento problem sa nevyriešil.

Pri CPU variante s meniacim sa počtom vlákien sa výrazne nemení rýchlosť overovania, naproti tomu pri GPU variante pri zdvojnásobení počtu GPU sa zdvojnásobí aj rýchlosť.

GPU varianta oproti CPU urýchlila proces obnovy hesla 35 násobne. Pri použití 4x GPU až 135 násobne. 8 miestne heslo vieme nájsť za 25 , 10 miestne za 12 dní.

9.3 RAR

V tomto module je nutné vykonať veľké množstvo kôl hašovacej funkcie pre overenie hesla a s tým aj priamo závisí rýchlosť.

9.3.1 RAR 5

Pre overenie hesla je vo východzom režime nutné vykonať 32 768 kôl funkcie PBKDF2-HMAC-SHA256, ktorá samotný proces obnovy hesla značne spomalí. Ako môžeme vidieť v tabuľke 9.6, tak rýchlosť overovanie hesiel je mnohonásobne nižšia ako pri OpenSSL module. Ale narozdiel od OpenSSL modulu, môžeme pozorovať zvýšenú rýchlosť pri použití ďalšieho vlákna. Pri zdvojnásobení počtu vlákien sa rýchlosť zdvojnásobí.

Pri GPU variante sa podarilo urýchliť obnovu oproti CPU 43-násobne (vid 9.7).

Ale pri použití ďalšej grafickej karty pri krátkom hesle neznamená nárast výkonu, niekedy nastal opačný efekt, kedy pri použití ďalších grafických kariet sa výkon znížil. Ako

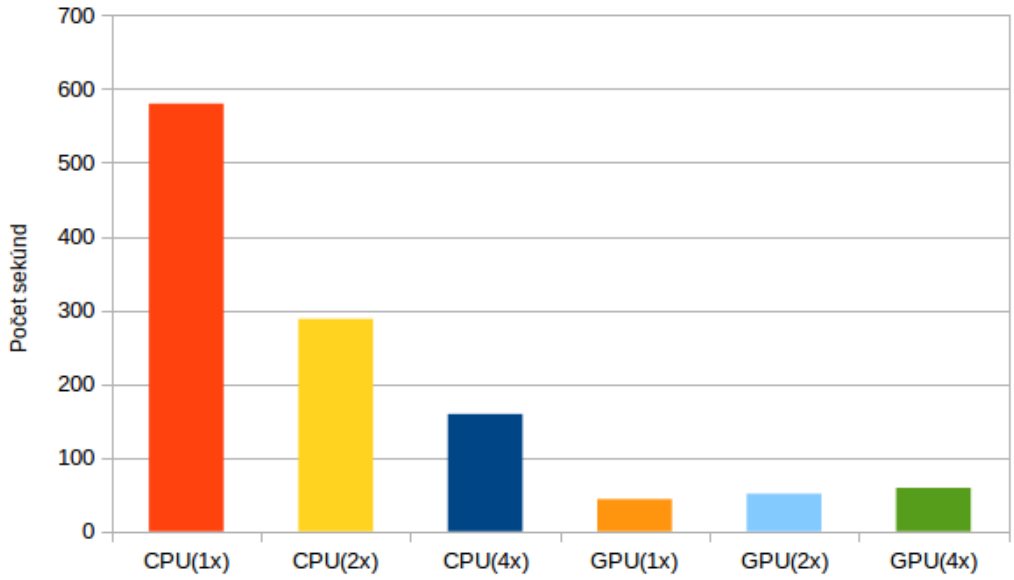
	1 Vlákno	2 Vlákna	4 Vlákna
Rýchlosť	33	65	130

Tabuľka 9.4: Rýchlosť overovania hesla pri formáte RAR5 CPU varianta

	1x GPU	2x GPU	4x GPU
Rýchlosť	1381	2505	5603

Tabuľka 9.5: Rýchlosť overovania hesla pri formáte RAR5 GPU varianta

môžeme vidieť na grafe 9.4 pri obnove 3 miestneho hesla, sa pri použití ďalšej GPU čas pre nájdenie hesla o pár sekúnd zvýšil, to iste nastalo aj pri 4x GPU.



Obr. 9.4: Porovnanie času potrebného pre nájdenie 3 miestneho hesla

9.3.2 RAR 3

Pre overenie jedného hesla je potrebné vykonať 262 144 kôl hašovacej funkcie SHA-1. Aj pri vyššom množstve kôl, oproti RAR5 je obnova hesla pri RAR3 rýchlejšia kvôli rozdielnej hašovacej funkcie.

	1 Vlákno	2 Vlákna	4 Vlákna
Rýchlosť	114	228	432

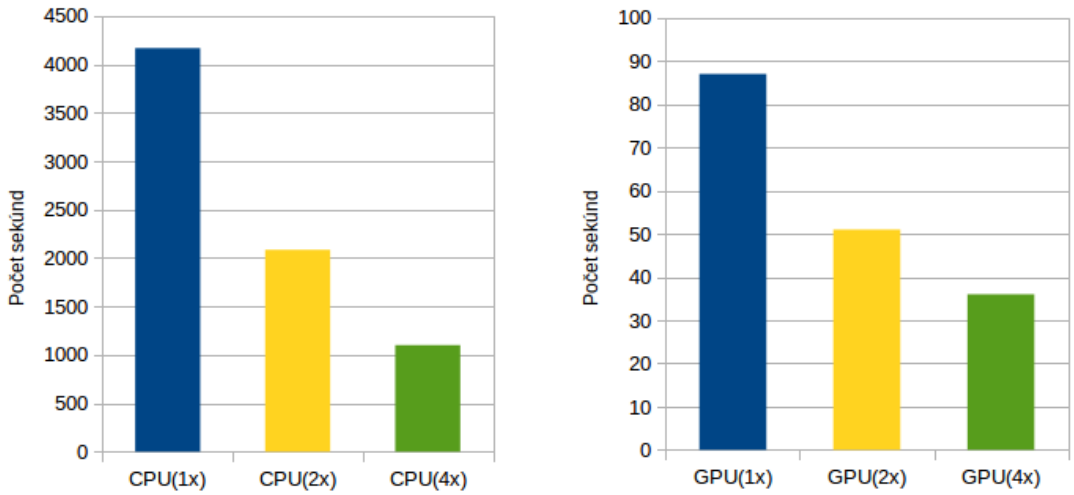
Tabuľka 9.6: Rýchlosť overovania hesla pri formáte RAR3 CPU varianta

Pri GPU variante, v porovnaní jedného CPU oproti jednému GPU nastalo 48-násobne zrýchlenie. Zaznamenali sme len 2.5-násobný zvýšený výkon pri 4x GPU oproti 1x GPU.

Na grafe 9.5 môžeme vidieť čas potrebný pre nájdenie 4 miestneho hesla. Pri použití jedného vlákna zistíme heslo za 1 hodinu a 9 minút, naproti tomu pri použití 4 grafických kariet, vieme tento čas zredukovať na 36 sekúnd.

	1x GPU	2x GPU	4x GPU
Rýchlosť	5462	9318	13201.

Tabuľka 9.7: Rýchlosť overovania hesla pri formáte RAR5 GPU varianta



Obr. 9.5: Čas potrebný pre obnovu 4 miestneho hesla

9.3.3 Porovnanie nástrojov

Konkurenčné nástroje dosahujú lepších výsledkov. Algoritmy vo Wrathione neboli zoptimalizované. Ako môžeme vidieť v tabuľke 9.8 Wrathion je približne o 17% rýchlejší pri CPU variante RAR3, ale pri GPU variante je skoro 6-krát pomalší.

Pri RAR5 (vid 9.9) je už aj pri CPU variante 5-krát pomalší a pri GPU až 22-násobne pomalší.

	CPU 1-vlákno	CPU 2-vlákna	CPU 4-vlákna	GPU
Wrathion	114	228	432	5462
John The Ripper	95	190	370	30068
oclHashcat	#	#	#	10011

Tabuľka 9.8: Porovnanie rýchlosti nástrojov pri RAR3

	CPU 1-vlákno	CPU 2-vlákna	CPU 4-vlákna	GPU
Wrathion	33	65	130	1381
John The Ripper	165	320	650	30025
oclHashcat	#	#	#	16181

Tabuľka 9.9: Porovnanie rýchlosti nástrojov pri RAR5

Nástroje John the Ripper a oclHashcat dosahujú niekoľko násobne vyššie rýchlosti pri GPU, ako náš nástroj Wrathion. Z tohoto môžme vidieť, že ostáva ešte veľa priestoru na optimalizácie a zvýšenie rýchlosti. Pri nástroji oclHashcat nebola meraná CPU varianta, keďže dokáže využívať len GPU.

Komerčný nástroj Advanced Archive Password Recovery [1] od Elcomsoft zaostával za všetkými nástrojmi. Nedokáže spracovať RAR5 archív a ani využiť GPU. Pri CPU variante nebolo možné nastaviť počet vlákien a využíval ich všetkých 12 ale len na 10%. Pri RAR3 archíve dosahoval rýchlosti len 200 hesiel za sekundu, takúto rýchlosť dosahuje Wrathion už pri použití 2 vlákien.

[1] https://www.elcomsoft.com/archpr.html

Kapitola 10

Záver

Snaha o dešifrovanie súboru bez znalosti hesla pomocou útoku hrubou silou je časovo náročne. Použitie grafickej karty pre akceleráciu výpočtov urýchli proces obnovy hesla, ale záleží na danom formáte, ako veľmi je túto úlohu možné zrýchliť. V implementácii šifrovania formátov sú použité hašovacie funkcie s veľkým množstvom kôl, ktoré zvyšujú potrebný čas pre proces obnovy hesla.

Pri šifrovaní je používaná náhodná vygenerovaná soľ, ktorá je uložená aj v samotnom zašifrovanom súbore. Soľ sa kombinuje s heslom, najčastejšie konkatenáciou a výsledok je predaný ako vstup do hašovacej funkcie. Je nutné zo súboru prečítať soľ a pri každom overení hesla aplikovať soľ na heslo, ktoré sú vstupom do hašovacieho algoritmu. Použitie soli znemožňuje použitie tabuľky predpočitaných hašov, ktorá by proces obnovy hesla zrýchlila.

Implementácia modulu je špecifická záležitosť, keďže formáty sa od seba výrazne líšia. Každý formát ma svoj systém pre uchovanie dát, jeho kompresiu, šifrovanie a overenie validnosti hesla. Vymenované veci sa nelíšia len medzi formátmi, ale aj v rámci jedného formátu inej revízie. Preto je nutne pre každú revíziu každého formátu mať osobitný modul.

Modul RAR implementuje čítanie RAR archívu a následný proces obnovy hesla pre RAR5 a RAR3 so zašifrovanou hlavičkou. Nepodporovaný je formát RAR3 bez zašifrovanej hlavičky, kedy je nutné vykonať aj dekompresiu a porovnať kontrolný súčet súboru. Pre tento typ bolo už implementované vytiahnutie potrebných dát, vrátane nájdenie najmenšieho súboru v archíve.

Modul OpenSSL implementuje čítanie súboru a vytiahnutie soli. Umožňuje obnoviť heslo pri šifrovacom algoritme AES a formátoch GZIP a BZIP2. Modul bol navrhnutý tak aby sa ľahko dal rozšíriť o ďalšie šifrovacie algoritmy tak aj o ďalšie formáty. Modul sa dá v budúcnosti upraviť, aby bol použiteľný aj pre súbory, ktoré boli zašifrované iným programom ako OpenSSL, napríklad VeraCrypt.

Rýchlosť implementovaných modulov je pomalšia ako u konkurenčných nástrojov, ktoré sú vyvíjané dlhší čas a väčšou skupinou ľudí a tým sú aj lepšie optimalizované. Moduly v nástroji Wrathion boli implementované podľa špecifikácii daných formátov a šifier. Neboli využité žiadne optimalizácie. Zostáva ešte priestor pre optimalizáciu jednotlivých hašovacích funkcii a kryptografických algoritmov, ktoré môžu urýchliť celý proces obnovy hesla.

Literatúra

[1] DAEMEN, J.; RIJMEN, V.: *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002, ISBN 978-3-662-04722-4.

[2] DEUTSCH, P.; GAILLY, J.-L.; ADLER, M.; aj.: GZIP file format specification version 4.3. RFC 1952, May 1996, [online]. [cit. 2016-05-15]. Dostupné z: http://www.rfc-editor.org/rfc/rfc1952.txt.

[3] GUANG, H.; JIANHUA, M.; BENXIONG, H.: Password Recovery for RAR Files Using CUDA. In *Dependable, Autonomic and Secure Computing, 2009. DASC '09. Eighth IEEE International Conference on*, Institute of Electrical & Electronics Engineers (IEEE), Prosinec 2009, s. 486–490.

[4] HANDSCHUH, H.: *Encyclopedia of Cryptography and Security*, kapitola SHA-0, SHA-1, SHA-2 (Secure Hash Algorithm). Boston, MA: Springer US, 2011, ISBN 978-1-4419-5906-5, s. 1190–1193.

[5] HRANICKÝ, R.; MATOUŠEK, P.; VESELÝ, V.; aj.: Experimental Evaluation of Password Recovery in Encrypted Documents. In *Proceedings of ICISSP 2016*, Roma: SciTePress - Science and Technology Publications, 2016, ISBN 978-989-758-167-0, s. 299–306.

[6] KALISKI, B.: PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, September 2000, [online]. [cit. 2016-05-15]. Dostupné z: http://www.rfc-editor.org/rfc/rfc2898.txt.

[7] MOUHA, N.: *Encyclopedia of Cryptography and Security*, kapitola MD4-MD5. Boston, MA: Springer US, 2011, ISBN 978-1-4419-5906-5, s. 768–771.

[8] MUNSHI, A.: Technická zpráva, 2012-11-14, [online]. [cit. 2016-05-15]. Dostupné z: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf.

[9] PRENEEL, B.: *Encyclopedia of Cryptography and Security*, kapitola HMAC. Boston, MA: Springer US, 2011, ISBN 978-1-4419-5906-5, s. 559–560.

[10] ROSHAL, A.: Technická zpráva, 2015, [online]. [cit. 2016-05-15]. Dostupné z: http://www.rarlab.com/technote.htm.

[11] SCHMIED, J.: *GPU akcelerované prolamování šifer*. Diplomová práce, 2014.

[12] SEWARD, J.: Technická zpráva, 2008-09-27, [online]. [cit. 2016-05-15]. Dostupné z: http://www.bzip.org.

Prílohy

Príloha A

Obsah CD

Na elektronický nosič boli priložené následujúce súčasti bakalárskej práce:

- **src/** – Zdrojové súbory nástroja Wrathion
- **latex/** – Latex súbory pre technickú správu
- **testfiles/** – Testovacie súbory pre implementované moduly
- **sprava.pdf** – PDF verzia technickej správy
- **files.txt** – Textový dokument obsahujúci pridané a zmenené súbory v nástroji
- **README** – Textový dokument obsahujúci informácie ako spustiť nástroj