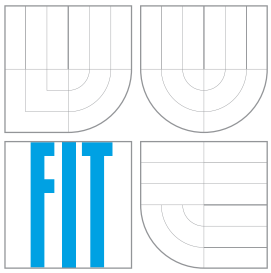
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYUŽITÍ HEURISTIK PŘI OBNOVĚ HESEL POMOCÍ GPU

USE OF HEURISTICS FOR PASSWORD RECOVERY WITH GPU ACCELERATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETER GAZDÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK HRANICKÝ
SUPERVISOR

BRNO 2016

Abstrakt

Tato práce se zabývá různými technikami, které umožňují zlepšení procesu obnovy hesel akcelerovaného pomocí GPU. V první části je představen Markovův model a jednoduché regulární výrazy, které umožňují výrazně redukovat stavový prostor generovaných hesel. Tyto techniky jsou založeny na pozorování hesel tvořených uživateli. Byl navržen paralelní algoritmus, který kombinuje tyto techniky. Závěr práce obsahuje výsledky experimentů, které dokazují výhody použití Markovova modelu.

Abstract

This thesis discusses various techniques to enhance the password recovery process with GPU acceleration. The first part introduces a Markov model and simple regular expressions. These techniques dramatically reduce the password space to be searched. This is based on observations of users and their use of letters in passwords. We propose the design of a parallel algorithm that combines both techniques. Last part of the thesis contains the results of experiments to prove benefits of Markov model.

Klíčová slova

obnova hesel, Markovův model, kryptografie, GPU, GPGPU, OpenCL

Keywords

Password Recovery, Markov Model, Cryptography, GPU, GPGPU, OpenCL

Citace

Peter Gazdík: Využití heuristik při obnově hesel pomocí GPU, bakalářská práce, Brno, FIT VUT v Brně, 2016

Využití heuristik při obnově hesel pomocí GPU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Hranického.

.....
Peter Gazdík
18. května 2016

Poděkování

Rád bych touto cestou poděkoval Ing. Radkovi Hranickému za odbornou pomoc, cenné rady a vstřícnost při konzultacích.

Obsah

1 Úvod **3**

2 OpenCL **4**
2.1 Model platformy . . . 4
2.2 Exekučný model . . . 4
2.3 Pamäťový model . . . 5
2.4 Programový model . . . 5

3 Nástroj Wrathion **6**
3.1 Architektúra . . . 6
3.2 Detekcia formátu a šifrovania . . . 6
3.3 Generovanie hesiel . . . 7
3.3.1 Brute-force generátor . . . 7
3.3.2 Slovníkový generátor . . . 7
3.4 Overovanie hesiel . . . 8
3.4.1 CPU implementácia . . . 8
3.4.2 GPU implementácia . . . 8

4 Techniky pre zdokonaľovanie útokov **11**
4.1 Markovské modely . . . 11
4.1.1 Markovský model prvého rádu . . . 12
4.1.2 Vrstvový Markovský model . . . 13
4.2 Maskovanie . . . 13

5 Návrh generátora hesiel **15**
5.1 Generátor založený na Markovských modeloch . . . 15
5.1.1 Frekvenčná analýza . . . 15
5.1.2 Dátové štruktúry . . . 16
5.1.3 Generovanie hesla . . . 16
5.1.4 Inkrementácia dĺžky hesla . . . 17
5.1.5 Rezervácia hesiel . . . 19
5.2 Generátor využívajúci maskovanie . . . 20
5.2.1 Syntax . . . 20
5.2.2 Generovanie hesla . . . 21

6 Implementácia **22**
6.1 Získavanie štatistík . . . 22
6.1.1 Formát výstupného súboru . . . 22

6.1.2 Architektúra aplikácie 23
6.2 Generátor hesiel 24
6.2.1 Inicializácia dátových štruktúr 24
6.2.2 Rezervácia hesiel 24
6.2.3 Generovanie hesiel 24
6.3 Riadenie generovania a overovania hesiel 25

7 Experimenty **27**
7.1 Meranie úspešnosti pri obnove hesiel 27
7.1.1 Postup experimentu 27
7.1.2 Výsledky experimentu 28
7.2 Meranie výkonu 29
7.2.1 Testovacia zostava 29
7.2.2 Postup experimentu 29
7.2.3 Výsledky experimentu 30

8 Záver **33**

A Obsah CD **35**

B Kompletné výsledky experimentov **36**

Kapitola 1

Úvod

Užívateľmi vytvárané heslá predstavujú v súčastnosti hlavný spôsob zabezpečenia. Kvôli zvýšeniu bezpečnosti pristupujú užívatelia k voľbe dlhších hesiel, prípadne do nich zahŕňajú číslice a špeciálne znaky. Pri prekročení určitej dĺžky následne zlyhávajú metódy obnovy hesiel založené na útoku hrubou silou. Útok hrubou silou (brute-force útok) spočíva vo vyskúšaní všetkých prípustných permutácií znakov nad danou abecedou. Z tohoto dôvodu je nevyhnutné pristúpiť k metódam generovania, ktoré využívajú toho, že ľudia vytvárajú svoje heslá s ohľadom na zapamätateľnosť.

Hlavným faktorom pre dobrú zapamätateľnosť hesla je fonetická podobnosť so slovami v nejakom prirodzenom jazyku. Dosiahnutie tejto vlastnosti pri strojovo generovaných heslách umožňuje Markovský model, ktorý je bežne využívaný v oblasti spracovania prirodzeného jazyka.

Druhou vlastnosťou, ktorou sa heslá vyznačujú, je výskyt špeciálnych znakov a číslic. Pozorovaním používaných hesiel však môžeme vidieť, že tieto znaky sa v hesle nenachádzajú úplne náhodne a je tak možné určiť často používané vzory. Pre generovanie hesiel zodpovedajúcich týmto vzorom je možné využiť maskovanie, ktoré je založené na jednoduchých regulárnych výrazoch.

Zmysel tejto práce je bližšie analyzovať tieto metódy pre použitie pri generovaní a obnove hesiel pomocou GPU. Okrem analýzy obsahuje taktiež návrh paralelného algoritmu pre dosiahnutie tohoto cieľa a popis jeho implementácie v podobe rozšírenia nástroja Wrathion. V závere práce je implementované riešenie podrobené experimentom, ktoré dokazujú použiteľnosť týchto techník pri procese obnovy hesiel.

Práca je rozdelená do niekoľkých kapitol. V kapitole 2 je popísaný framework OpenCL vrátane bližšieho popisu jeho architektúry. Nasleduje kapitola 3, ktorá popisuje architektúru nástroja Wrathion z pohľadu jeho rozšírenia o nový druh generátora. Kapitola 4 predstavuje techniky pre zlepšenie generovania hesiel založených na frekvenčnej analýze, menovite Markovské modely a techniku maskovania. Kapitola 5 potom obsahuje návrh algoritmu, ktorý využíva tieto techniky. Kapitola 6 popisuje implementáciu navrhnutého algoritmu v prostredí OpenCL kvôli akcelerácii na GPU. Použiteľnosť a efektivita implementovaného nástroja je ukázaná v kapitole 7.

Kapitola 2

OpenCL

OpenCL (*Open Compute Language*) je framework, ktorý umožňuje programovanie paralelných výpočtov nezávisle na konkrétnej hardwarovej platforme. Jeho vývoj zastrešuje priemyslové konzorcium Khronos [5], do ktorého patria spoločnosti ako napr. Apple, AMD, Intel. Programovanie pozostáva z dvoch častí: programovanie hostiteľskej aplikácie v jazyku C alebo C++, ktorá je vykonávaná na procesore a programovanie tzv. kernelov v jazyku OpenCL C, ktoré bežia na zariadeniach podporujúcich OpenCL.

Architektúru OpenCL je možné popísať pomocou nasledujúcich modelov: model platformy, pamäťový model, exekučný model, programový model. Bližšie sú tieto modely rozobraté v nasledujúcich podkapitolách.

2.1 Model platformy

Model platformy (*platform model*) definuje vysokoúrovňovú reprezentáciu heterogénnej platformy kompatibilnej s OpenCL. Pozostáva z nasledujúcich častí:

- hostiteľ (*host*) je podporný systém, ktorý umožňuje beh OpenCL aplikácie. Najčastejšie úlohu hostiteľa zabezpečuje procesor,
- zariadenie (*OpenCL device*), na ktorom sú spúšťané OpenCL aplikácie (kernely). Hostiteľ môže byť pripojený k viacerým zariadeniam (GPU, CPU, ...),
- každé zariadenie pozostáva z výpočetných jednotiek (*compute units*),
- výpočetné jednotky sa ďalej skladajú z výpočetných elementov (*processing elements*).

2.2 Exekučný model

Vykonávanie programu je rozdelené na dve časti. Prvá časť pozostáva z programu hostiteľa a druhá časť je program pre zariadenia, ktorý je tvorený kolekciou tzv. kernelov.

Inštancia kernelu sa nazýva pracovná jednotka a má svoj jednoznačný identifikátor (*global ID*). Jednotlivé pracovné jednotky sú združované do pracovných skupín. Každá pracovná skupina má svoj identifikátor (*work-group ID*) a každá pracovná jednotka má ešte identifikátor v rámci pracovnej skupiny (*local ID*).

Vlastné spúšťanie kernelov na zariadeniach je riadené pomocou rady príkazov (*command-queue*), čo sú vlastne objekty, kde sú uložené OpenCL príkazy. Viacej týchto rád umožňuje

zariadeniu vykonávať paralelne viacej príkazov bez potreby synchronizácie. Príkazy v radách môžu byť vykonávané buď *in-order* alebo *out-of-order*. *In-order* predstavuje sériové vykonávanie príkazov v rade, pričom predchádzajúci príkaz musí byť ukončený, než je spustený ďalší. V prípade *Out-of-order* vykonávania sa nečaká na dokončenie predchádzajúceho príkazu, čo potom zväčša vyžaduje dodatočné synchronizačné príkazy.

2.3 Pamäťový model

V OpenCL je možné používať niekoľko druhov pamätí. Jednak sa tu nachádza *pamäť hostiteľa*, ktorá je prístupná len hostiteľovi a jednotlivé zariadenia do nej nemôžu pristupovať. *Pamäť zariadení* sa potom delí ešte na nasledovné časti:

- **Globálna pamäť** – hlavná pamäť, ktorá je prístupná všetkým pracovným jednotkám pre čítanie aj zápis.
- **Pamäť konštánt** – oblasť globálnej pamäte, ktorej obsah zostáva počas vykonávania programu nemenný.
- **Lokálna pamäť** – pamäť zdieľaná všetkými pracovnými jednotkami v rámci pracovnej skupiny. Je určená na zdieľanie dát medzi pracovnými jednotkami patriacimi do rovnakej pracovnej skupiny. Používa sa na zníženie počtu prístupov do globálnej pamäte, nakoľko je výrazne rýchlejšia.
- **Súkromná pamäť** – privátna pamäť prístupná len jednotlivým pracovným jednotkám, pričom pracovná jednotka môže pristupovať len do svojej súkromnej pamäte.

V rámci kernelu nie je možné vykonávať dynamickú alokáciu. To je umožnené len hostiteľovi pred spustením výpočtu na zariadení.

2.4 Programový model

OpenCL podporuje ako dátový, tak aj úlohový paralelizmus, prípadne ich kombináciu. Dátový paralelizmus predstavuje spúšťanie pracovných jednotiek s rovnakým kódom, ktoré spracovávajú spoločný balík dát. Naopak, úlohový paralelizmus je postavený na rozdelení problému do niekoľkých úloh, ktoré budú vykonávané paralelne. V OpenCL je možné na jednom zariadení vykonávať viacej rôznych kernelov. Tieto kernely potom môžu spolupracovať a predávať si výstupy medzi sebou, pričom sa synchronizujú pomocou OpenCL udalostí.

Kapitola 3

Nástroj Wrathion

Táto kapitola obsahuje popis nástroja Wrathion, ktorý vznikol v rámci diplomovej práce Jana Schmieda [8] a v súčasnej dobe prebieha jeho ďalší vývoj v rámci projektu zameraného na boj s počítačovou kriminalitou.

Wrathion je multiplatformový nástroj určený k obnove hesiel dokumentov umožňujúci akceleráciu na GPU. V aktuálnej verzii podporuje formáty súborov DOC, PDF a ZIP. Vďaka modulárnej architektúre je možné jeho jednoduché rozšírenie o podporu ďalších formátov a ich verzií.

Všetky časti nástroja sú implementované v jazyku C++ okrem zdrojového kódu pre grafické karty, ktorý je v jazyku OpenCL C, ktorého popis nájdete v kapitole 2.

3.1 Architektúra

Architektúru aplikácie bližšie zobrazuje obrázok č. 3.1. Aplikácia je koncipovaná do nasledujúcich častí:

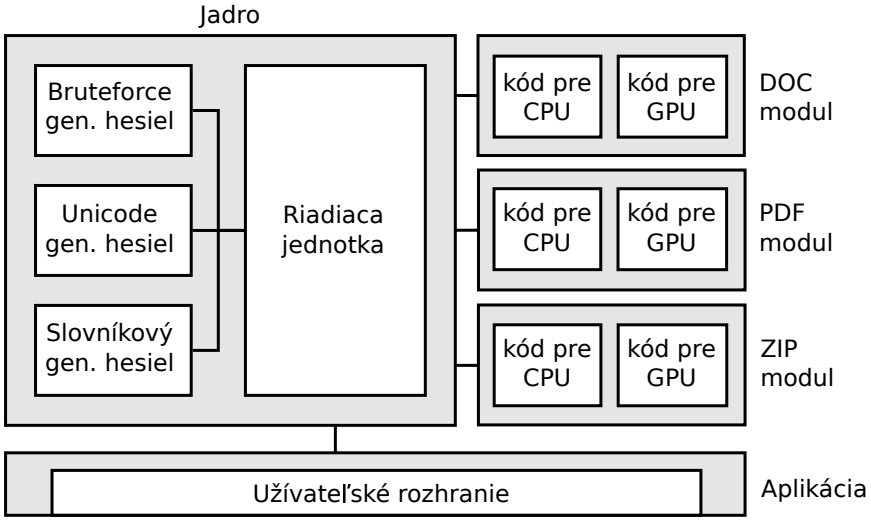
- **Jadro** – obsahuje generátory hesiel a zavádzač jednotlivých modulov
- **Moduly** – obsahujú program pre spustenie na CPU aj GPU
- **Aplikácia** – inicializuje jadro a poskytuje jednoduché textové rozhranie

3.2 Detekcia formátu a šifrovania

Prvou úlohou, ktorú nástroj vykonáva, je detekcia typu dokumentu a použitého šifrovania. Vo väčšine prípadov je určenie typu súboru možné na základe podpisu[1]. Podpis je jednoznačný reťazec, ktorý sa zvyčajne nachádza na začiatku súboru.

Potom, čo je identifikovaný typ súboru, môže dôjsť k detekcii použitého šifrovania. Nakoľko je táto detekcia úzko spätá s daným formátom, implementujú ju jednotlivé moduly samostatne. Dochádza teda k zavedeniu zodpovedajúceho modulu a ten analýzou metadát určí použité šifrovanie.

[1] File Signatures – http://www.filesignatures.net/



Obr. 3.1: Architektúra Wrathionu [2]

3.3 Generovanie hesiel

Generátor hesiel je podstatnou súčasťou nástroja. Jeho úlohou je generovanie hesiel, ktoré budú následne overené. V súčasnosti nástroj využíva generátory dvoch typov:

- **Brute-force generátor** – vytvára všetky možné kombinácie reťazcov nad znakovou sadou,
- **Slovníkový generátor** – heslá získava z vopred pripraveného slovníka s často používanými heslami.

3.3.1 Brute-force generátor

Brute-force generátor hesiel vytvára všetky možné permutácie nad danou abecedou. S výhodou je využívaný na sofistikovanejšie heslá, ktoré nie je možné prelomiť slovníkovým útokom, viď sekciu 3.3.2. S narastajúcou dĺžkou hesla narastá exponenciálne aj čas potrebný na jeho generovanie. Prelomenie hesla nad určitú dĺžku teda nie je týmto druhom generátora v prijateľnom čase dosiahnuteľné.

Generátor je však jednoducho paralelizovateľný a veľmi dobre vertikálne aj horizontálne škálovateľný. Teda zväčšovanie výkonnosti je možné jednak v rámci jedného počítača pridávaním výpočetných jednotiek, ale aj pridaním viacerých uzlov do výpočetného klastra. Toto je možné implementovať využitím knižnice OpenMPI. Princíp brute-force generátora hesiel je bližšie popísaný v sekcii 3.4.

3.3.2 Slovníkový generátor

Slovníkový útok je veľmi jednoduchý a umožňuje prelomenie hesla vo veľmi krátkom čase, teda v prípade, že je toto heslo obsiahnuté v slovníku. Vychádza z toho, že veľké množstvo používaných hesiel sú jednoduché slová jazyka alebo jednoducho zapamätateľné sekvencie znakov, ako napr. 'abcdefgh', '123456789' atď.
Dáta slovníka sú uložené v súbore, kde sa na každom riadku nachádza jedno heslo. Načítavanie hesiel zo súboru je však oproti rýchlosti overovania hesiel výrazne pomalšie a spomaľuje

tak rýchlosť celého procesu. Nakoľko však slovníky neobsahujú veľké množstvo záznamov, nie je to veľkou prekážkou.

3.4 Overovanie hesiel

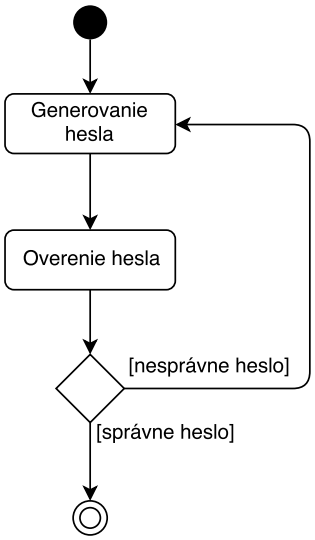
Overovanie hesiel je poslednou časťou procesu obnovy hesla. Verifikácia je špecifická pre každý typ dokumentu, verziu a použitý šifrovací algoritmus.

Algoritmus verifikácie pozostáva z výpočtu hešovacej a šifrovacej funkcie. Výsledok výpočtu je potom porovnaný s hodnotou hešu uvedenou v dokumente. Ak sa hodnoty zhodujú, proces obnovy hesla týmto končí.

Ďalej budú v tejto podkapitole popísané rôzne scenáre overovania hesla v závislosti na tom, či generovanie alebo overovanie prebieha na CPU alebo GPU.

3.4.1 CPU implementácia

Pri implementácii overovania na CPU nie je problém vygenerovať heslo a ihneď ho aj overiť. Schematické znázornenie činnosti je zrejmé z obrázku č. 3.2. Zdrojom generovaných hesiel môže byť v tomto prípade brute-force generátor alebo slovníkový generátor.



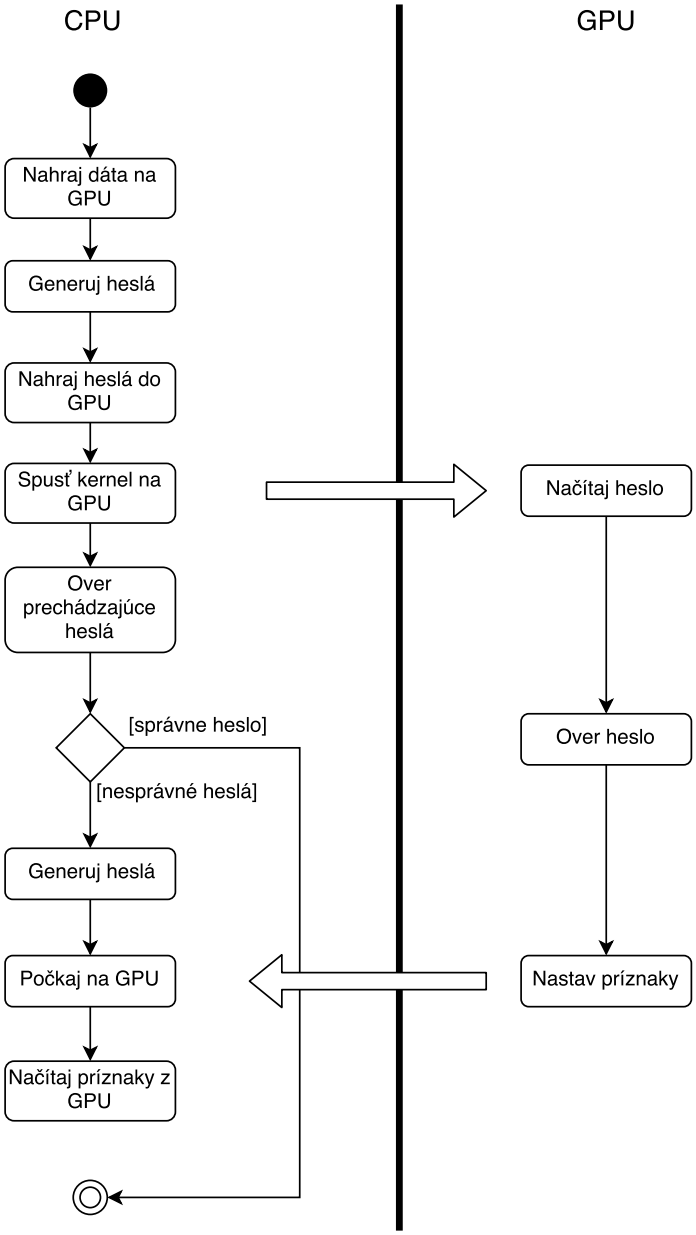
Obr. 3.2: Princíp generovania a obnovy hesla na CPU [8]

3.4.2 GPU implementácia

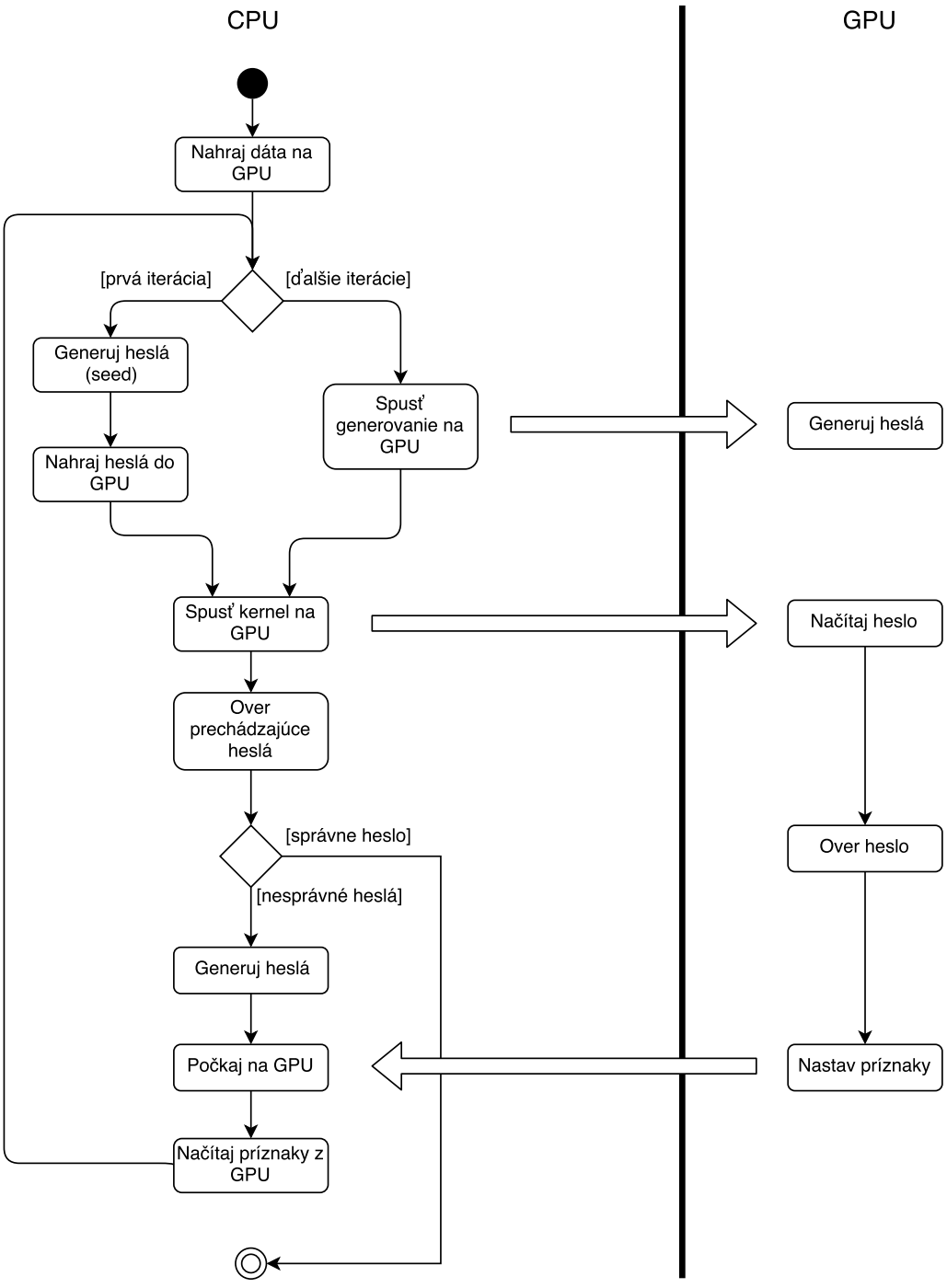
Činnosť programu pri využívaní GPU je o niečo zložitejšia. Musíme totiž zaistiť nahranie dát do pamäte GPU, spustenie kernelu a nahranie výsledkov späť do operačnej pamäte systému.

Okrem toho je potrebné v GPU alokovať pamäť o veľkosti postačujúcej na celú dobu behu programu, nakoľko inštancie na GPU nemôžu vykonávať dynamickú realokáciu pamäte. Pamäť je nastavená na dĺžku $globalWorkSize \cdot (maxPasswordLength + 1)$, kde $globalWorkSize$ predstavuje počet inštancií OpenCL kernelu a $maxPassowrdLength + 1$ je miesto pre generované heslo spolu s jednobajtovou informáciou o aktuálnej dĺžke hesla.

Schéma na obrázku 3.3 znázorňuje generovanie hesiel na CPU a overovanie na GPU. Generovanie hesiel v tomto prípade môže zabezpečovať brute-force alebo slovníkový generátor. V okamihu, keď aj generovanie prebieha na grafickej karte, ako to popisuje obrázok 3.4, môže ísť len o brute-force generátor.



Obr. 3.3: Princíp generovania hesiel na CPU a overovanie na GPU [8]



Obr. 3.4: Princíp generovania aj overovania na GPU [8]

Kapitola 4

Techniky pre zdokonaľovanie útokov

Táto kapitola popisuje metodológie generovania hesiel, ktoré generujú heslá efektívnejším spôsobom než je tomu pri brute-force generátore.

Ako bolo už spomenuté v úvode, užívateľmi vytvárané heslá predstavujú v dnešnej dobe stále hlavný spôsob zabezpečenia. Kvôli zvýšeniu bezpečnosti pristupujú užívatelia k voľbe dlhších hesiel, prípadne do nich zahŕňajú číslice a špeciálne znaky [1]. Pri prekročení určitej dĺžky následne zlyhávajú metódy obnovy hesiel založené na útoku hrubou silou. Okrem toho sa v súčasnej dobe používajú stále zložitejšie šifrovacie algoritmy, napr. Bcrypt [10], ktoré nahrádzajú štandardné hešovacie funkcie. Takéto algoritmy vyžadujú niekoľkonásobne viacej času na výpočet a znižujú tak rýchlosť overovania.

Efektívnosť generovania sa tak ukazuje ako kľúčová pre úspešné prelomenie hesla. Aj keď by bolo najvhodnejšie pokryť všetky možné permutácie hesiel a mať tak nástroj, ktorý zaručene uspeje, z časového hľadiska to pri dlhých heslách nie je dosiahnuteľné. Z tohoto dôvodu je nutné vyskúšať efektívnejšie postupy ako prvé a až v prípade neúspechu siahnuť po tých menej efektívnych, ktorých výstupom je väčšia množina jedinečných hodnôt hesiel.

Jedným z takýchto postupov je aj využitie slovníkového generátora, ktorý využíva ako zdroj databázu najpoužívanejších hesiel alebo slov niektorého jazyka, ktoré sa na mieste hesla taktiež môžu vyskytovať. Tento postup má však značne nízku úspešnosť a nebudeme sa ním v tejto práci ďalej zaoberať.

Všetky nižšie popísané optimalizačné metódy spojuje využívanie frekvenčnej analýzy.

4.1 Markovské modely

Markovský model je matematický nástroj, ktorý je bežne používaný v spracovaní prirodzeného jazyka [7] a je základom systémov pre rozpoznávanie reči. Na prelamovanie hesiel ho prvýkrát aplikovali Arvind Narayanan a Vitaly Schmatikov [6]. Niekoľko ďalších prác [3] [4] sa zaoberá analýzou rôznych druhov Markovských modelov a ich úspešnosti.

Využitie tohoto modelu pri generovaní hesiel je založené na pozorovaní, že ľudia tvoria svoje heslá tak, že ich znaky sledujú skrytý Markovský model. To znamená, že pravdepodobnosť, že n-tý znak hesla je x, je daná na základe znaku na pozícii $n-1$.

Potom pravdepodobnosť výskytu nejakého hesla je súčinom pravdepodobností výskytov všetkých jeho znakov, ktoré ho tvoria. Názornejšie je to možné vidieť na nasledujúcom príklade

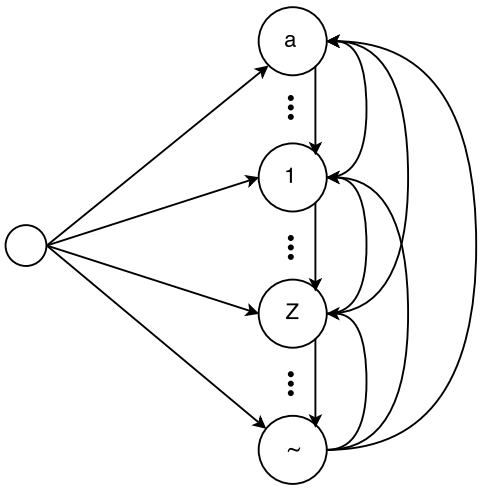
$$P(\text{heslo}) = P(\text{h}) \cdot P(\text{e}|\text{h}) \cdot P(\text{s}|\text{e}) \cdot P(\text{l}|\text{s}) \cdot P(\text{o}|\text{l}), \tag{4.1}$$

kde $P(s)$ je pravdepodobnosť výskyty reťazca s a $P(x|y)$ je pravdepodobnosť výskytu znaku x po znaku y, pričom $x, y \in s$. Hodnota každého $P(x|y)$ je získaná z frekvenčnej analýzy na dostatočne veľkom slovníku [4].

Všeobecne existujú aj Markovské modely vyšších rádov, pri ktorých je pravdepodobnosť výskytu znaku x daná funkciou viacerých znakov, ktoré mu predchádzajú. Hoci tieto modely dosahujú lepšie výsledky, ich implementácia je rádovo pomalšia [6] a slovník, z ktorého je model vytváraný vyžaduje enormne veľké množstvo vzoriek, aby bol model vôbec použiteľný. Preto sa táto práca bude ďalej zameriavať len na Markovské modely prvého rádu.

4.1.1 Markovský model prvého rádu

Markovský model pozostáva z uzlov a ohodnotených orientovaných hrán medzi uzlami. Generovanie hesla začína v počiatočnom stave z ktorého vedie stochastický prechod do všetkých uzlov, ktorých počet je daný počtom znakov znakovej sady z ktorej je heslo zložené. Uzly v hlavnej vrstve majú potom analogicky prechody medzi sebou. V každom prechode je vygenerovaný jeden znak hesla. Proces generovania končí pri dosiahnutí požadovanej dĺžky hesla. Ak je aktuálny stav daný len hodnotou jedného znaku a nie dvoch a viacerých znakov, hovoríme o Markovskom modeli prvého rádu. Príklad takéhoto modelu popisujúceho heslo zložené z ASCII znakov je možné vidieť na obrázku č. 4.1.

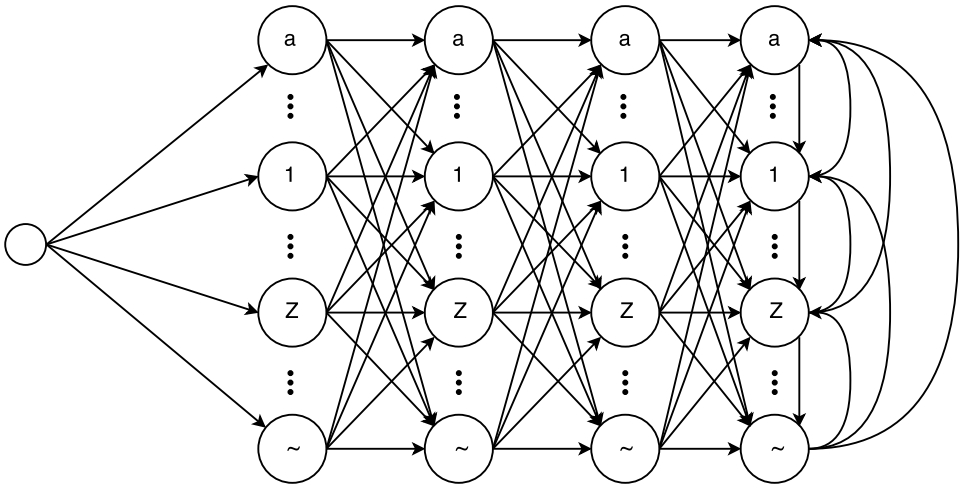


Obr. 4.1: Príklad Markovského modelu prvého rádu [9]

K vytvoreniu modelu je najskôr potrebné analyzovať distribúciu znakov v nejakom známom slovníku. Týmto slovníkom môže byť buď databáza prelomených hesiel alebo slovník nejakého jazyka. V prípade použitia slovníka niektorého jazyka je však nevýhodou, že obsahuje len abecedné znaky a neumožňuje potom generovať heslá obsahujúce číslice a špeciálne znaky.

4.1.2 Vrstvový Markovský model

Vrstvový Markovský model pozostáva z n vrstiev tvorených Markovskými modelmi prvého rádu, kde n je dĺžka hesla, ktoré chceme vygenerovať. Pravdepodobnosti prechodov medzi stavmi vo vrstvách i a $i+1$ sú určené frekvenčnou analýzou dvojíc znakov i a $i+1$. Na obrázku 4.2 je možné vidieť model so 4 vrstvami, ktorý popisuje heslo o dĺžke štyroch ASCII znakov.



Obr. 4.2: Príklad vrstvového Markovského modelu [9]

Tento model je schopný zachytiť väčšie množstvo informácii o distribúcii znakov v hesle. Vhodný je najmä v prípadoch, kedy je frekvenčná analýza vykonávaná nad slovníkom pozostávajúceho z uniknutých hesiel a my sme tak schopný zachytiť napr. výskyt číslic na určitých pozíciach v hesle. Tento predpoklad potvrdzuje aj práca W. Tanseyho [9], ktorý v niektorých prípadoch dosiahol až 15% zlepšenia pri použití 8-vrstvového modelu v porovnaní s Markovským modelom prvého rádu.

4.2 Maskovanie

Stavový priestor hesiel, ktorý by pozostával len zo sekvencie znakov nedosahuje príliš dobré pokrytie. Ľudia totiž pri tvorbe hesiel často používajú kombináciu veľkých a malých písmen, resp. čísel a špeciálnych znakov, nakoľko sú k tomu vo väčšine prípadov navádzaní, resp. obmedzení daným systémom zabezpečenia. Napriek tomu distribúcia znakov výsledného hesla má stále ďaleko od náhodného rozloženia.

Nižšie je možné vidieť niekoľko často používaných vzorov, ktoré sa v takýchto heslách vyskytujú:

- v alfanumerických heslách sa číslice vo väčšine prípadov nachádzajú na konci reťazca,
- prvý znak sekvencie je v porovnaní s ostatnými znakmi výrazne častejšie veľké písmeno,
- pomer malých písmen k veľkým je vo väčšine prípadov výrazne vyšší.

Vyjadrenie takýchto podmienok je možné využitím konečných stavových automatov, ktoré budú popisovať regulárne výrazy popisujúce štruktúru hesla. Tento prístup býva označovaný ako maskovanie.

Počet metaznakov regulárneho výrazu je vhodné voliť s ohľadom na zachytenie používaných vzorov získaných frekvenčnou analýzou.
Maskovanie v súčasnosti implementuje veľké množstvo nástrojov na prelamovanie hesiel, ako John the Ripper [1], Hashcat [2]. Maskovanie je možné kombinovať s inými optimalizačnými technikami, ako napr. s Markovským modelom.

[1] John the Ripper – http://www.openwall.com/john/
[2] Hashcat – http://hashcat.net/

Kapitola 5

Návrh generátora hesiel

Kapitola sa zaoberá návrhom generátora založeného na metódach rozobraných v predchádzajúcej kapitole. Cieľom pri návrhu je nájsť taký algoritmus, ktorý bude možné spúšťať paralelne bez nutnosti vzájomnej synchronizácie medzi jednotlivými inštanciami. Požiadavkou je teda algoritmus, ktorý pre daný index i vráti i-tý reťazec reprezentujúci heslo, ktorý spĺňa všetky nami požadované vlastnosti.

5.1 Generátor založený na Markovských modeloch

5.1.1 Frekvenčná analýza

Nevyhnutnou súčasťou generátora založeného na Markovských modeloch je matica pravdepodobnostných prechodov medzi jednotlivými stavmi. Príklad takejto matice pre znakovú sadu pozostávajúcu zo znakov a–z je možné vidieť na obr. 5.1. Hodnoty $p_{x,y}$ udávajú pravdepodobnosti prechodu zo stavu x do stavu y, pričom $x, y \in \{\varepsilon, \mathrm{a}, \mathrm{b}, \ldots, \mathrm{z}\}$, kde ε predstavuje prázdny reťazec. Súčet pravdepodobností v riadku matice musí byť rovný 1.

$$P = \begin{pmatrix} p_{\varepsilon,\mathrm{a}} & p_{\varepsilon,\mathrm{b}} & \cdots & \cdots & p_{\varepsilon,\mathrm{z}} \\ p_{\mathrm{a},\mathrm{a}} & p_{\mathrm{a},\mathrm{b}} & \cdots & \cdots & p_{\mathrm{a},\mathrm{z}} \\ p_{\mathrm{b},\mathrm{a}} & p_{\mathrm{b},\mathrm{b}} & \cdots & \cdots & p_{\mathrm{b},\mathrm{z}} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ p_{\mathrm{z},\mathrm{a}} & p_{\mathrm{z},\mathrm{b}} & \cdots & \cdots & p_{\mathrm{z},\mathrm{z}} \end{pmatrix}$$

Obr. 5.1: Príklad matice pravdepodobnostných prechodov

Pre získanie hodnôt pravdepodobností je potrebné vykonať frekvenčnú analýzu nad nejakým slovníkom. Frekvenčná analýza je založená na skúmaní počtu dvojíc x, y, kde $x, y \in \{\Sigma, \varepsilon\}$ a Σ predstavuje použitú znakovú sadu. Na základe týchto počtov sa následne určia jednotlivé hodnoty pravdepodobností. Pri znakoch vyskytujúcich sa na prvej pozícii v reťazcoch skúmaného slovníka hodnota x predstavuje prázdny reťazec ε. V prípade analýzy pre potreby vrstvového Markovského modelu je nutné vytvoriť matice pravdepodobnostných prechodov pre každú pozíciu znaku v reťazci zvlášť, viď sekciu 4.1.2.

V praxi sa ukazuje, že aj napriek použitiu relatívne veľkého slovníka (15 mil. hesiel) existuje veľké množstvo dvojíc x, y, ktoré sa v danom slovníku vôbec nevyskytujú. To znamená, že pre niektoré konkrétne x existuje veľké množstvo znakov y, ktorých pravdepodobnosť výskytu po znaku x je rovná nule. Ak však urobíme analýzu výskytu znakov y v danom slovníku, zistíme, že veľké množstvo týchto znakov sa na užívateľmi tvorených heslách podieľa.

Z tohoto dôvodu je vhodné určiť pravdepodobnosť výskytu znakov y v danom slovníku a na základe týchto pravdepodobností dodatočne upraviť nulové pravdepodobnosti pri dvojiciach x, y tak, aby pôvodné hodnoty pravdepodobností stanovené Markovským modelom boli stále vyššie, ako dodatočne upravené pravdepodobnosti. Implementácia takejto úpravy je popísaná v kapitole 6.1.2.

5.1.2 Dátové štruktúry

Na to, aby sme dokázali efektívne generovať heslá s využitím Markovského modelu, potrebujeme previesť maticu pravdepodobnostných prechodov na vhodnejšiu formu dátovej štruktúry. Pri generovaní nás nezaujímajú samotné hodnoty pravdepodobností, ale len to, ktoré stavy budú po aktuálnom stave nasledovať a v akom poradí. Preto môžeme túto maticu previesť na dvojrozmerné pole, ktorého indexy riadkov budú zodpovedať aktuálnym stavom a hodnoty v týchto riadkoch stavom, ktoré po aktuálnom stave nasledujú, zoradené zostupne podľa pravdepodobnosti. Na obr. 5.2 môžeme vidieť príklad takejto štruktúry pre znakovú sadu zloženú zo znakov a–z.

ε	n	p	s	z	v	...
a	n	l	t	v	c	...
b	o	e	a	i	r	...
c	h	i	e	k	o	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮
z	a	e	o	i	n	...

Obr. 5.2: Dvojrozmerné pole nahradzujúce maticu pravdepodobnostných prechodov

5.1.3 Generovanie hesla

K dosiahnutiu dobrej paralelizovateľnosti musí byť algoritmus generovania navrhnutý s ohľadom na čo najmenší počet synchronizácií medzi jednotlivými inštanciami. V ideálnom prípade teda požadujeme algoritmus, ktorý vygeneruje nové heslo len na základe zadaného indexu. Odlišnosť indexu jednotlivých inštancií je zaistená pridelením rôznej hodnoty pred začatím výpočtu.

Okrem indexu vyžaduje algoritmus užívateľom zadanú hodnotu prahu, pomocou ktorej sme schopní obmedziť stavový priestor všetkých generovaných hesiel. Prah vlastne vyjadruje počet kombinácii znakov, ktoré sa môžu na danej pozícii hesla vyskytnúť. V kontexte s upravenou maticou pravdepodobnostných prechodov sa jedná o počet stĺpcov tejto matice, ktoré sú pre generovanie použité. V prípade, že by prah nebol použitý, algoritmus by generoval celý stavový priestor hesiel ako je tomu u brute-force generátora, len v inom poradí.

Algoritmus 1 Generovania jedného hesla s využitím Markovského modelu

Vstupy: *index*, *markovCharSet*[ ][ ], *passLength*, *threshold*
Výstup: *outBuffer*[ ]

```
1: currentCharSet = markovCharSet[ε]
2: for i = 0 to passLength do
3:     partialIndex ← index mod threshold        ▷ Určenie indexu pre aktuálny riadok
4:     index ← index ÷ threshold                 ▷ Zvyšok pre indexovanie ďalších riadkov

5:     char ← currentCharSet[partialIndex]       ▷ Znak na aktuálnej pozícii
6:     currentCharSet ← markovCharSet[char]      ▷ Riadok matice pre ďalšiu iteráciu

7:     outBuffer[i] ← char                       ▷ Zápis do bufra
8: end for
9: index ← index
```

Algoritmus 1 popisuje princíp generovania jedného hesla o dĺžke *passLength* na základe zadaného indexu s využitím upravenej matice pravdepodobnostných prechodov zo sekcie 5.1.1. V každej iterácii cyklu dochádza ku generovaniu jedného znaku hesla, ktorého výber je realizovaný z jedného riadku dvojrozmerného poľa znakov *markovCharSet*, pričom sa z tohoto riadku uvažuje vždy len počet stĺpcov daný hodnotou prahu *threshold*. Výber riadku z tejto matice je realizovaný na základe aktuálne vygenerovaného znaku, pričom pri prvej iterácii je k výberu použitá hodnota prázdneho reťazca, viď sekciu 5.1.1. Výpočet čiastkového indexu pre výber znaku na aktuálnej pozícii v hesle je realizovaný na 3. riadku. Na 4. riadku dochádza k určeniu zvyšku, ktorý je použitý pri výpočte v ďalších iteráciách. Počet iterácií je daný dĺžkou aktuálne generovaného hesla.

Vrstvový Markovský model

Pre potreby vrstvového Markovského modelu je potrebné pre každú pozíciu hesla použiť rozdielnu maticu pravdepodobnostných prechodov. V analógii s dátovou štruktúrou *markovCharSet* použitou v algoritme 1 sa jedná o pridanie jednej dimenzie, čím dostávame trojrozmerné pole. Generovanie jedného hesla s využitím tohoto modelu popisuje algoritmus 2. Zároveň je pri generovaní použitá odlišná hodnota prahu pre každú pozíciu, ktorá je uložená v poli *thresholds*.

Tento algoritmus je zároveň možné použiť pre generovanie hesiel pomocou klasického Markovského modelu, kedy postačuje dvojrozmerné pole nakopírovať na všetky pozície v trojrozmernom poli. Výhodou takéhoto prístupu je aj jednoduché rozšírenie o podporu maskovania, viď sekcia 5.2.

5.1.4 Inkrementácia dĺžky hesla

Bežný postup prelamovania hesiel je založený na testovaní rôznych dĺžok hesiel, pričom sa obvykle začína heslami o dĺžke jedna a táto dĺžka sa postupne inkrementuje. Pri využití algoritmu 1 by bolo najskôr nutné vygenerovať všetky heslá pre konkrétnu dĺžku, následne inkrementovať túto dĺžku a znova započať generovanie od nultého indexu. Tento postup by však komplikoval proces rezervácie hesiel, viď sekcia 5.1.5, nakoľko by bolo obtiažne stanoviť index pre novú dĺžku hesla.

Algoritmus 2 Generovania jedného hesla s využitím vrstvového Markovského modelu

Vstupy: *index*, *markovCharSet*[ ][ ][ ], *passLength*, *thresholds*[ ]
Výstup: *outBuffer*[ ]

```
1: currentCharSet = markovCharSet[0][ε]
2: for i = 0 to passLength do
3:     partialIndex ← index mod thresholds[i]      ▷ Určenie indexu pre aktuálny riadok
4:     index ← index ÷ threshold                   ▷ Zvyšok pre indexovanie ďalších riadkov

5:     char ← currentCharSet[i][partialIndex]      ▷ Znak na aktuálnej pozícii
6:     currentCharSet ← markovCharSet[char]        ▷ Riadok matice pre ďalšiu iteráciu

7:     outBuffer[i] ← char                         ▷ Zápis do bufra
8: end for
```

Z tohoto dôvodu je vhodnejšie zaviesť tzv. globálny index, ktorý bude využívaný k rezervácii hesiel a pre potreby generovania sa jeho hodnota prevedie na lokálny index a zároveň sa z neho odvodí aj aktuálna dĺžka hesla. K realizácii prevodu bude nutné predpočítať pre nejakú dĺžku hesla n počet indexov, ktoré boli použité ku generovaniu hesiel s dĺžkou 1 až $n-1$ a následne odčítaním tejto hodnoty od globálneho indexu dostaneme lokálny index pre heslo s dĺžkou n, ktorý predstavuje index používaný v algoritme 1 Počet indexov potrebných pre vygenerovanie hesiel s danou dĺžkou je daný ako variácia s opakovaním vyjadrená ako

$$V(k, n) = n^k, \tag{5.1}$$

kde n predstavuje dĺžku hesla a k hodnotu prahu. V prípade odlišnej hodnoty prahu pre každú pozíciu v hesle je potrebné vzťah vyjadriť vo forme súčinu

$$\prod_{i=1}^{n} k_i, \tag{5.2}$$

kde k_i predstavuje hodnotu prahu na pozícii i a n dĺžku hesla. Výpočet počtu indexov pre všetky dĺžky bude daný vzťahom

$$\sum_{j=1}^{n} \prod_{i=1}^{j} k_i. \tag{5.3}$$

Nakoľko je maximálna dĺžka hesla pred generovaním pevne stanovená, sme schopní tieto hodnoty predpočítať pre všetky možné dĺžky n. Tieto hodnoty potom uložíme do jednorozmerného poľa, ktorého prvý prvok bude predstavovať počet indexov pre dĺžku 0, t.j. hodnotou 0 a nasledovať budú hodnoty od 1 po maximálnu dĺžku, ktorá bude generovaná. Príklad takéhoto poľa pre maximálnu dĺžku hesla 7 znakov pri hodnote prahu 5 pre všetky pozície je možné vidieť v tabuľke 5.1.

Dĺžka	0	1	2	3	4	5	6	7
Počet indexov	0	5	30	155	780	3905	19530	97655

Tabuľka 5.1: Pole s predpočítanými počtami indexov

Algoritmus 3 popisuje prevod globálneho indexu *globalIndex* na dĺžku *length* a lokálny index *localIndex* s využitím poľa predpočítaných hodnôt *numOfIndexes*. V prvom kroku algoritmu dochádza k určeniu dĺžky porovnávaním globálneho indexu s predpočítanými hodnotami až dovtedy, kým je hodnota globálneho indexu väčšia alebo rovná hodnote pre aktuálne stanovenú dĺžku. Pomocou takto stanovenej dĺžky určíme lokálny index odčítaním predpočítanej hodnoty pre dĺžku $length - 1$.

Algoritmus 3 Prevod globálneho indexu na lokálny index a dĺžku

Vstupy: $globalIndex, numOfIndexes$
Výstupy: $localIndex, length$

```
1: length = 1
2: while globalIndex >= numOfIndexes[length] do
3:     length = length + 1                          ▷ Určenie aktuálnej dĺžky
4: end while

5: localIndex = globalIndex − numOfIndexes[length − 1]          ▷ Prevod indexu
```

5.1.5 Rezervácia hesiel

Najjednoduchší spôsob rozdelenia hesiel medzi jednotlivé inštancie generátorov je založený na stanovení indexov jednotlivým inštanciám pred začatím výpočtu. Následne každá inštancia zvyšuje index o krok daný počtom všetkých inštancií. Tento postup je nevhodný v prípade, kedy sa jednotlivé inštancie odlišujú v rýchlosti generovania. Môže tak totiž nastať situácia, kedy generovanie správneho hesla pripadne na generátor s najpomalšou rýchlosťou.

Preto je vhodnejšie pre rozdeľovanie hesiel medzi jednotlivé inštancie využiť tzv. rezerváciu hesiel. Postup spočíva v rezervovaní určitého balíka hesiel pre každú inštanciu generátora. Generátor potom generuje heslá s krokom 1 až do doby, než nevygeneruje všetky heslá z balíčka. Následne si rezervuje nový balíček hesiel. Princíp rezervácii je bližšie priblížený v algoritme 4, ktorý je prebratý z pôvodnej práce o nástroji Wrathion [8, str. 30]. K zabezpečeniu výlučného prístupu k zdieľanej hodnote indexu *sharedIndex* je použitý zámok reprezentovaný premennou *mutex*. V kritickej sekcii algoritmu dochádza k uloženiu aktuálneho stavu zdieľaného indexu a jeho zvýšenie o hodnotu *reservationSize*. Okrem samotnej

Algoritmus 4 Rezervácia balíka hesiel

Vstupy: $reservationSize, sharedIndex, lastIndex$
Výstupy: $reservedIndex, reserveredSize$

```
1: MutexLock(mutex)
2: reservedIndex = sharedIndex
3: sharedIndex = sharedIndex + reservationSize
4: MutexUnlock(mutex)

5: if reservedIndex + reservationSize > lastIndex then
6:     reservedSize = lastIndex − reservedIndex
7: else
8:     reservedSize = reservationSize
9: end if
```

rezervácie je potrebné stanoviť veľkosť rezervovaného balíka $reservedSize$. V prípade, že je táto hodnota menšia alebo rovná nule, došlo k vygenerovaniu všetkých hesiel a je potrebné zastaviť generovanie.

Tabuľka 5.2 zobrazuje niekoľko krokov paralelného generovania s využitím rezervácie. Na začiatku si každý z troch generátorov rezervuje balík indexov o veľkosti 10 z ktorého generuje heslá. Po ich vyčerpaní si rezervuje nový balík indexov. Generovanie končí, keď už nie je možné rezervovať ďalšie heslá.

Generátor x		**Generátor** y		**Generátor** z	
index	**heslo**	**index**	**heslo**	**index**	**heslo**
0	h_0	10	h_{10}	20	h_{20}
1	h_1	11	h_{11}	21	h_{21}
⋮	⋮	⋮	⋮	⋮	⋮
9	h_9	19	h_{19}	29	h_{29}
30	h_{30}	40	h_{40}	50	h_{50}
31	h_{31}	41	h_{41}	51	h_{51}
⋮	⋮	⋮	⋮	⋮	⋮

Tabuľka 5.2: Ukážka paralelného generovania hesiel s využitím rezervácie

5.2 Generátor využívajúci maskovanie

Pri implementácii maskovania sa obmedzíme na podmnožinu regulárnych výrazov, ktoré pozostávajú len z konkrétnych znakov znakovej sady alebo z metaznakov, ktoré zastupujú nejakú podmnožinu použitej znakovej sady. Nebudeme teda uvažovať použitie kvantifikátorov, ktoré by špecifikovali počet výskytov znaku alebo metaznaku.

5.2.1 Syntax

Syntax regulárnych výrazov je prebratá z konkurenčných nástrojov Hashcat a JohnTheRipper. Cieľom je dosiahnuť jednoduchšiu použiteľnosť pre užívateľov, ktorí tieto nástroje poznajú. Zoznam jednotlivých metaznakov je uvedený v tabuľke 5.3. Mataznaky pozostávajú vždy zo znaku `?`, za ktorým nasleduje alfanumerický znak, ktorý bližšie špecifikuje jeho význam. Pre uvedenie znaku `?` ho je potrebné uviesť dvakrát.

Metaznak	**Popis**
`?l`	malé písmená
`?u`	veľké písmená
`?d`	číslice
`?s`	špeciálne znaky
`?a`	kombinácia všetkých predchádzajúcich, t.j. `?l`, `?u`, `?d`, `?s`

Tabuľka 5.3: Popis jednotlivých metaznakov

5.2.2 Generovanie hesla

Algoritmus generovania s využitím maskovania sa nijak nelíši od algoritmu 2 pre vrstvový Markovský model. Jediný rozdiel je pri vytváraní dvojrozmerných polí obsahujúcich znaky pre generovanie. Z každého poľa je potrebné odstrániť všetky znaky, ktoré nevyhovujú zadanej maske a prípadne znížiť hodnotu prahu o počet odstránených znakov.

Kapitola 6

Implementácia

Táto kapitola popisuje implementáciu generátora hesiel pre nástroj Wrathion na základe návrhu uvedeného v kapitole 5. V rámci tohoto popisu budú uvedené len časti, ktoré majú súvislosť s procesom generovania hesiel. Popis ostatných častí nástroja Wrathion je možné nájsť v diplomovej práci Jana Schmieda [8]. Implementácia jednotlivých častí je realizovaná v jazyku C++ a OpenCL C pri programoch určených pre grafickú kartu.

6.1 Získavanie štatistík

Nevyhnutnou súčasťou pri generovaní hesiel pomocou Markovského modelu sú štatistiky vo forme matice pravdepodobnostných prechodov. K získaniu týchto štatistík je potrebné vykonať frekvenčnú analýzu nejakého slovníka uniknutých hesiel, viď kapitola 5.1.1. Nakoľko je však takáto analýza pomerne časovo náročná a veľkosť slovníka môže dosahovať až niekoľko stoviek megabajtov, je vhodnejšie túto analýzu vykonávať mimo nástroj pre prelamovanie hesiel. Z tohoto dôvodu bol pre získavanie štatistík implementovaný samostatný nástroj, ktorého výstupom je matica pravdepodobnostných prechodov vo forme binárnych dát.

6.1.1 Formát výstupného súboru

Formát výstupného súboru je navrhnutý s ohľadom na uchovanie rôznych druhov štatistík. V súčasnej dobe sú to štatistiky pre klasický a vrstvový Markovský model.

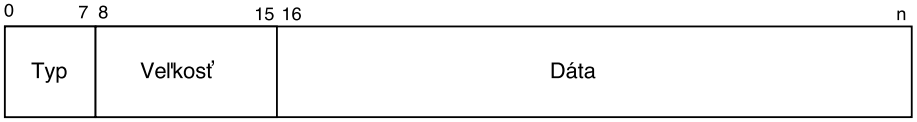
Na začiatku súboru sa nachádza textová hlavička, ktorá pozostáva z tzv. podpisu (File Signature), pomocou ktorého je možné detekovať použitie nevhodného súboru. Za podpisom nasledujú súhrnné informácie o štatistikách, ako je použité kódovanie znakovej sady a detailnejší popis štatistík. Hlavička je zakončená znakom s ASCII hodnotou 3. Príklad hlavičky je možné vidieť na obrázku 6.1.

```
%WSTAT-1.0%
\Encoding: ISO-8859-1
\Description: Subor obsahuje statistiky vytvorene zo slovnika
obsahujuceho uniknute hesla spolocnosti Rockyou v roku 2009
```

Obr. 6.1: Ukážka hlavičky súboru so štatistikami

Za textovou hlavičkou sa nachádzajú binárne dáta obsahujúce jednotlivé štatistiky. Aby bolo zabezpečené bezproblémové použitie súboru na rôznych architektúrach, sú jednotlivé bajty uložené ako big-endian, t.j. viac významný bajt je uložený ako prvý. K prevodu do big-endian sú využité funkcie `htonl` a `htons` z knižnice `arpa/inet.h` pod systémami typu UNIX, resp. `winsock2.h` pod systémom Windows. K spätnému prevodu potom slúžia funkcie `ntohl` a `ntohs`.

Blok binárnych dát pozostáva z jednobajtovej informácie o type daných štatistík, tzv. typovej signatúry. Nasleduje dvojbajtová informácia o veľkosti samotných štatistík v bajtoch a samotné dáta so štatistikami. Takýchto blokov dát môže súbor obsahovať niekoľko, maximálne však 256. Prirodzene nie je možné, aby súbor obsahoval viacej blokov s rovnakým typom, nakoľko by bolo nejednoznačné, ktoré štatistiky použiť pre generovanie.



Obr. 6.2: Blokové znázornenie formátu binárnych dát

6.1.2 Architektúra aplikácie

Základom celej aplikácie je abstraktná trieda `Statistics`, ktorá definuje jednoduché rozhranie pre ostatné triedy, ktoré implementujú vytváranie jednotlivých štatistík. V súčasnej dobe je implementované generovanie štatistík pre klasický a vrstvový Markovský model.

Klasický Markovský model

Vytváranie štatistík pre klasický Markovský model implementuje trieda `MarkovStatistics`. Umožňuje vytvoriť štatistiky nad slovníkom v ľubovoľnom 8-bitovom kódovaní, ako je ASCII, ISO-8859-1 a pod. Implementácia je založené na návrhu uvedenom v sekcii 5.1.1.

Základnou štruktúrou pri vytváraní štatistík je dvojrozmerné pole s 256 riadkami a 256 stĺpcami, ktoré je na začiatku inicializované samými nulami. Pri prechode slovníka sa každý výskyt dvojice znakov x, y pripočíta k aktuálnej hodnote v poli, pričom sa ako index riadku použije znak x a index stĺpca znak y. Takýmto spôsobom dostávame po prechode celého slovníka počty výskytov jednotlivých dvojíc. V tomto okamihu je možné na základe získaných hodnôt určiť maticu pravdepodobnostných prechodov pre Markovský model.

Za účelom spomínaného zlepšenia týchto štatistík je však ešte potrebný ďalší priechod slovníkom, pri ktorom dôjde k spočítaniu výskytov každého jedného znaku. Tieto počty výskytov sa ďalej prevedú na hodnoty v rozsahu 0 až 255. Takto získanými hodnotami sa následne upravia všetky nulové výskyty dvojíc x, y pričítaním počtu výskytov znaku y. Predtým je však nutné zvýšiť všetky nenulové výskyty x, y o hodnotu 256, aby nedošlo k narušeniu hodnôt získaných prvým prechodom.

Na záver sú počty výskytov prevedené na hodnoty pravdepodobností v rozsahu 0 až 1. Pre uloženie do súboru však čísla s plávajúcou rádovou čiarkou nie sú vhodné a preto je pri zápise do súboru vykonaný prevod na 16-bitové bezznamienkové čísla. Výstupná veľkosť takto vytvorených štatistík je $256 * 256 * 2 = 131072$ bajtov. Typová signatúra pri tomto druhu štatistík má hodnotu 1.

Vrstvový Markovský model

Implementácia sa nachádza v triede `LayeredMarkovStatistics`. Jediná odlišnosť oproti implementácii pre klasický Markovský model je vo vytváraní štatistík pre dvojice znakov x, y v závislosti na pozícii znaku x v práve analyzovanom reťazci. To znamená, že výstupom analýzy nie je len jedna matica pravdepodobnostných prechodov, ale 50 takýchto matíc, pričom hodnota 50 je pevne stanovená a mala by postačovať pre akýkoľvek vstupný slovník, nakoľko sa v súčasnosti takto dlhé heslá vyskytujú len zriedka.

Veľkosť vytvorených štatistík je $256 * 256 * 50 * 2 = 6553600$ bajtov a typová signatúra má hodnotu 2.

6.2 Generátor hesiel

K implementácii generátora hesiel bolo použité pôvodné rozhranie definované abstraktnou triedou `PassGen`. Pre potreby nového generátora však je nutné túto triedu rozšíriť o metódu `nextKernelStep` z dôvodu odlišného riadenia výpočtu na GPU. Okrem tejto úpravy sú zavedené nové parametre zadávané cez príkazovú riadku, ktoré sú špecifické pre generovanie hesiel s využitím Markovského modelu a maskovania.

6.2.1 Inicializácia dátových štruktúr

Ešte pred samotným procesom generovania hesiel je nevyhnutné vykonať inicializáciu dátových štruktúr špecifických pre tento druh generátora. Jedná sa predovšetkým o prevedenie matice pravdepodobnostných prechodov na dvojrozmerné pole znakov podľa postupu uvedeného v kapitole 5.1.2. Následne sú z tejto štruktúry vyradené znaky, ktoré nevyhovujú užívateľom zadanej maske. Celá inicializácia prebieha v metóde `initMemory`.

V prípade, že generovanie bude prebiehať na GPU, dochádza aj k inicializácii pamäte na grafickej karte v metóde `initKernel`.

6.2.2 Rezervácia hesiel

Pred samotným generovaním hesiel je potrebné rezervovať rozsah indexov pre generovanie. Túto rezerváciu vykonávajú jednotlivé vlákna rovnako pre generovanie na procesore aj grafickej karte. Implementácia sa nachádza v metóde `reservePasswords` a riadi sa algoritmom 4 popísaným v kapitole 5.1.5. Pri každom zavolaní dochádza k rezervácii počtu hesiel, ktoré je generátor schopný vygenerovať za jednu sekundu na základe aktuálnej rýchlosti generovania. V prípade generovania na GPU je tento počet upravený na najbližší násobok hodnoty GWS, aby bolo zabezpečené optimálne využitie grafickej karty.

6.2.3 Generovanie hesiel

Pri generovaní hesiel na CPU zabezpečuje vygenerovanie hesla metóda `getPassword` na základe algoritmu 2 uvedeného v kapitole 5.1.3. Po každom zavolaní metóda inkrementuje hodnotu indexu a v prípade, že došlo k vyčerpaniu rezervovaného rozsahu indexov, dochádza prostredníctvom metódy `reservePasswords` k aktualizácii hodnoty indexu.

Generovanie hesiel na GPU je riadené metódou `nextKernelStep`, ktorá zabezpečuje stanovenie indexov inštanciam generátorov bežiacich na grafickej karte, tzv. pracovným jednotkám (Work Items). Počet týchto indexov je daný veľkosťou GWS. Nakoľko by však prideľovanie indexu každej pracovnej jednotke bolo časovo náročné a taktiež by vyžadovalo

veľký prenos dát medzi operačnou pamäťou a pamäťou grafickej karty, dochádza len k stanoveniu tzv. offsetu. Tento offset nadobúda hodnotu indexu pre prvú pracovnú jednotku, pričom ostatné pracovné jednotky získajú hodnotu ich indexu pripočítaním hodnoty *globalID*, ktorá je v rozsahu 0 až GWS − 1 a je pre každú pracovnú jednotku unikátna, viď kapitolu 2 popisujúcu knižnicu OpenCL.

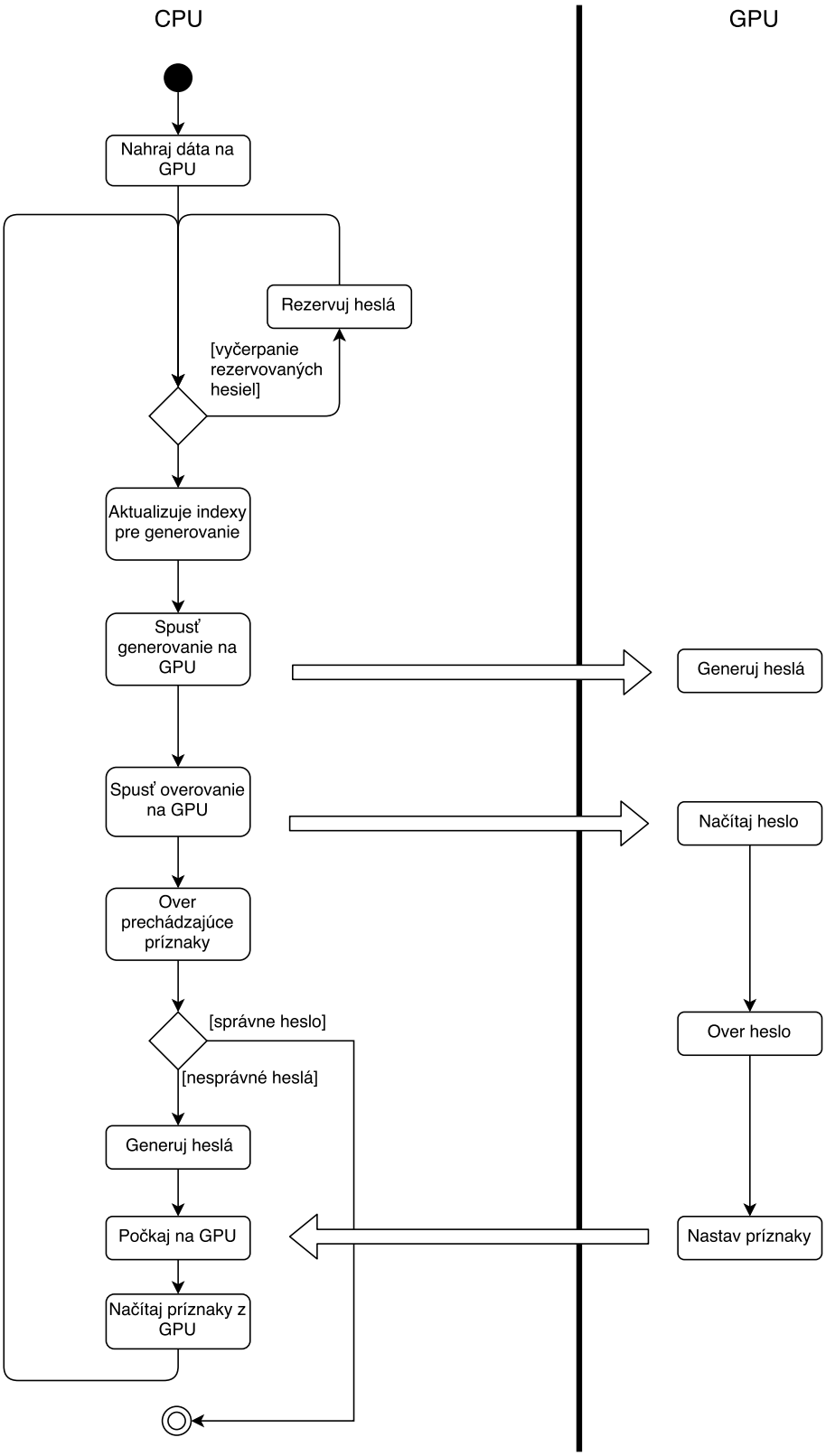
V jednom kroku teda dochádza na GPU k vygenerovaniu hesiel v celkovom počte daným hodnotou GWS. Heslá sa ukladajú do jednorozmerného poľa s počtom bajtov

$$\text{GWS} \cdot (maxPassLength + 1),$$

kde jedno heslo pozostáva okrem samotných znakov aj z jednobajtovej informácie o aktuálnej dĺžke hesla.

6.3 Riadenie generovania a overovania hesiel

Pri prelamovaní hesiel na GPU je proces generovania a následného overenia hesla zabezpečený triedou `GPUCracker`. Nakoľko sa však v pôvodnej implementácii nepočítalo s pridaním nového druhu generátora, bola nutná jej celková úprava. Tento krok však znemožnil používanie pôvodného generátora k prelamovaniu hesiel a pre zachovanie kompatibility je v budúcnosti potrebná úprava jeho implementácie. K výrazným zmenám došlo najmä v metóde `runThread`, ktorá zabezpečuje spúšťanie generátora hesiel na GPU a následne aj spúšťanie tzv. creckera, ktorý zabezpečuje overenie vygenerovaných hesiel. Celý proces znázorňuje diagram 6.3.



Obr. 6.3: Proces obnovy hesla pri generovaní a overovaní hesiel na GPU

Kapitola 7

Experimenty

Táto kapitola je venovaná experimentálnemu overeniu generátora hesiel navrhnutého v predchádzajúcej kapitole. V sekcii 7.1 sa nachádzajú merania úspešností, ktoré dosahuje implementovaný generátor pri redukovaní počtu generovaných hesiel. Sekcia 7.2 potom obsahuje porovnanie výkonu pri prelamovaní hesiel s využitím generátora založeného na Markovských modeloch v porovnaní s útokom hrubou silou.

7.1 Meranie úspešnosti pri obnove hesiel

Hlavnou výhodou generátora založeného na Markovských modeloch je možnosť obmedziť počet generovaných hesiel a výrazne tak znížiť čas potrebný pre prelomenie hesla. Na druhej strane však takáto redukcia znižuje šancu na nájdenie správneho hesla. Cieľom tohoto experimentu je overiť mieru úspešnosti v závislosti na redukcii generovaných hesiel. K meraniu bude využitá reprezentatívna vzorka hesiel.

7.1.1 Postup experimentu

Redukcia hesiel je závislá na zvolenej hodnote prahu, ktorý špecifikuje počet znakov, ktoré budú vyskúšané na každej pozícii v hesle. V prípade bežného útoku hrubou silou sú testované všetky platné znaky, čo predstavuje pri kódovaní ASCII celkovo 95 znakov, ktoré môžu byť použité. K výpočtu celkového počtu hesiel pre nejakú hodnotu dĺžky a prahu je použitý vzťah 5.3. Redukcia počtu hesiel je potom daná ako počet všetkých možných hesiel, ktoré je pri danej dĺžke možné vytvoriť, v pomere s celkovým počtom hesiel pri zadanej hodnote prahu. Takže napr. ak je počet všetkých možných hesiel 1000 a po redukcii bolo vygenerovaných 100 hesiel, jednalo sa o 10-násobnú redukciu. Pri meraní budeme hodnotu prahu postupne navyšovať. Okrem použitia rovnakých hodnôt prahu na každej pozícii budeme skúmať prípad, kedy je hodnota prahu na prvej pozícii vyššia oproti ostatným pozíciám a k zmene prahu bude dochádzať len na zvyšných pozíciach. Pri tomto postupe vychádzame z predpokladu, že človek je kreatívnejší pri výbere prvého znaku hesla ako pri ostatných znakoch, ktoré volí na základe tých predchádzajúcich.

Pre stanovenie úspešnosti porovnáme všetky vygenerované heslá s nejakou reprezentatívnou vzorkou užívateľských hesiel a stanovíme počet nájdených hesiel. Podiel počtu nájdených hesiel k počtu všetkých hesiel potom predstavuje úspešnosť. Okrem merania úspešnosti pri klasickom a vrstvovom Markovskom modeli budeme analyzovať aj kombinovanú úspešnosť týchto dvoch modelov pri rovnakej hodnote prahu. To znamená, že analyzujeme prienik nájdených hesiel týmito modelmi, podľa ktorého stanovíme unikátne heslá

pre každý model a pripočítame ich k tomuto prieniku.

Nakoľko však veľkosť vygenerovaných hesiel môže dosahovať až niekoľko gigabajtov, využívame k porovnávaniu špeciálne implementovaný nástroj, ktorý najskôr všetky heslá uloží do pamäte GPU vo forme hešovacej tabuľky. Bezprostredne po vygenerovaní každého hesla dochádza k overeniu, či sa v danej tabuľke nachádza. Po vygenerovaní všetkých hesiel sú vypísané nájdené heslá a ich počet. Nástroj je možné nájsť na priloženom CD.

Ako reprezentatívna vzorka hesiel nám budú slúžiť rôzne verejné datasety tvorené uniknutými užívateľskými heslami. Nakoľko potrebujeme tieto datasety aj k natrénovaniu Markovského modelu, budeme ich rôzne kombinovať tak, aby pre trénovanie bola použitá odlišná vzorka ako pre následný experiment.

7.1.2 Výsledky experimentu

Experimenty prebiehali na rôznych kombináciach datasetov. Nakoľko sa však zistenia pri jednotlivých kombináciach výrazne nelíšili, v následujúcom texte bude popísaná len jedna vybraná kombinácia. Výsledky ostatných experimentov môžete nájsť v prílohe B.

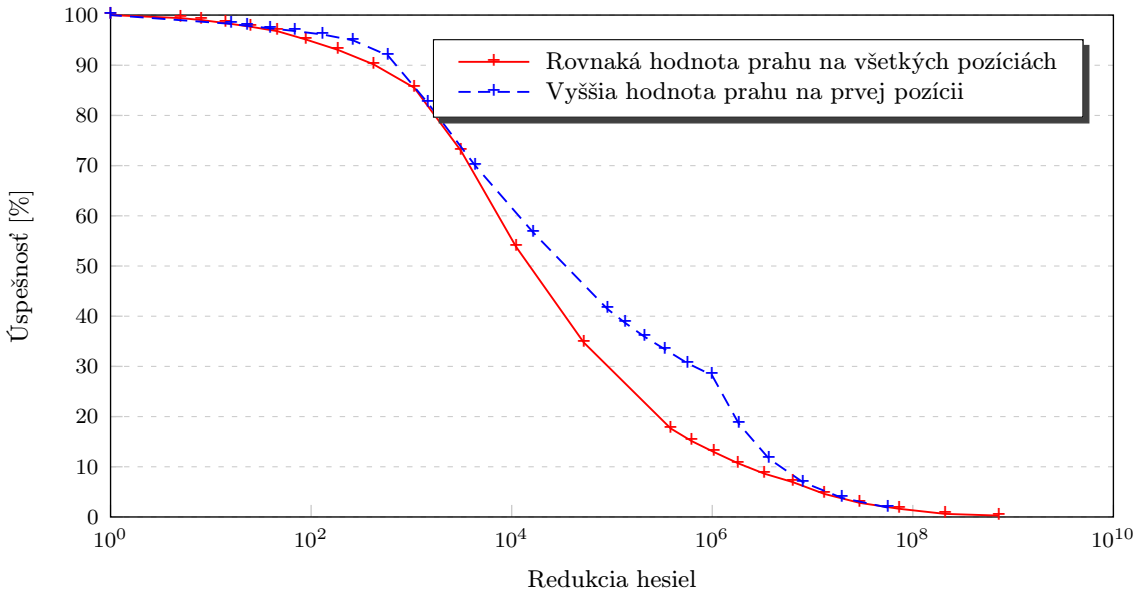
Pre vytvorenie štatistík boli použité heslá zo štyroch datasetov, nakoľko samostatne obsahovali pomerne malý počet hesiel. Konkrétne to boli uniknuté užívateľké heslá zo služieb Faithwriters, Hotmail, Myspace a PhpBB. Spolu obsahovali 238 811 hesiel. Na vyhodnotenie úspešnosti boli použité heslá s dĺžkou 1 až 7 znakov z datasetu Rockyou. Jednalo sa o 4 737 211 hesiel.

Graf 7.1 zobrazuje namerané výsledky pri použití klasického Markovského modelu. Na pohľad je z grafu zrejmá vyššia úspešnosť pri použití vyššej hodnoty prahu na prvej pozícii. V tabuľkách 7.1 a 7.2 môžeme vidieť porovnanie úspešností klasického a vrstvového Markovského modelu a ich kombinácie. Vrstvový model dosahoval o niečo lepšie výsledky. Výrazné navýšenie je potom vidieť pri skombinovaní obidvoch modelov.

Namerané výsledky dokazujú dobrú využiteľnosť Markovského modelu pre generovanie hesiel. Dosahované hodnoty úspešností v závislosti na redukcii sa všeobecne nelíšia pre iné dĺžky hesiel, čo je možné vidieť na meraniach nachádzajúcich sa v prílohe B.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	6 351 898	6,94	7,32	8,68
13	1 038 234	12,93	13,60	15,56
15	385 531	17,63	18,55	21,03
20	52 381	34,75	35,13	39,16
25	11 101	53,77	54,75	58,12
35	1 066	85,52	85,48	86,12
45	185	93,06	92,32	93,50
60	25	97,67	97,40	98,09
75	5	99,39	99,36	99,53

Tabuľka 7.1: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla



Obr. 7.1: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu

7.2 Meranie výkonu

Dôležitým faktorom pre použiteľnosť generátora je jeho rýchlosť. V tejto sekcii sa preto zameriame na vyhodnotenie rýchlosti nami implementovaného generátora v porovnaní s pôvodnou implementáciou brute-force generátora. Porovnávaná bude ich CPU aj GPU implementácia.

7.2.1 Testovacia zostava

CPU: Intel Core i7-5930K @ 3.50 GHz
RAM: 32 GB
GPU: 4x AMD Radeon R9 Fury X
OS: Windows 8.1 x64

Pri meraní výkonu na GPU budeme využívať len tri grafické karty, nakoľko jedna karta je primárne využívaná na vykresľovanie obrazu, čo by mohlo výrazne skresliť výsledky merania. Rovnako budeme postupovať pri CPU meraní, kde využijeme maximálne štyri procesorové jadrá zo šiestich dostupných.

7.2.2 Postup experimentu

Pre porovnanie rýchlosti obidvoch generátorov by bolo najjednoduchšie vyradiť fázu overovania vygenerovaných hesiel. Toto by však malo za následok výrazné skreslenie výsledkov, nakoľko pri bežnom použití pri prelamovaní hesla na GPU je využívaná doba, počas ktorej dochádza k overeniu hesiel na GPU, k inicializácii parametrov pre ďalšie generovanie. Preto budeme pri porovnávaní rýchlostí generátorov vychádzať z celkovej rýchlosti, ktorá zahŕňa aj overenie hesla.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,7	8 034 272	6,92	7,61	10,11
64,9	1 844 478	18,54	19,37	24,15
64,12	338 536	33,38	34,48	37,89
64,15	90 359	41,41	43,12	46,91
64,20	16 369	56,69	57,37	62,70
64,30	1 462	82,45	82,70	86,89
64,40	262	94,46	94,09	95,66
64,55	39	97,37	96,77	97,85
64,60	23	97,85	97,59	98,29

Tabuľka 7.2: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla

K tomuto účelu použijeme zašifrovaný PDF súbor s bezpečnostnou revíziou 5, pri ktorom je dosahovaná najvyššia rýchlosť prelamovania spomedzi podporovaných formátov. Vďaka tomu by mal byť pri tomto formáte najviac patrný vplyv generátora na výslednú rýchlosť. Okrem toho budeme meranie vykonávať aj na DOC dokumente vo verzii 97, ktorý má vyššiu úroveň zabezpečenia a rýchlosť overovania je pri ňom výrazne nižšia.

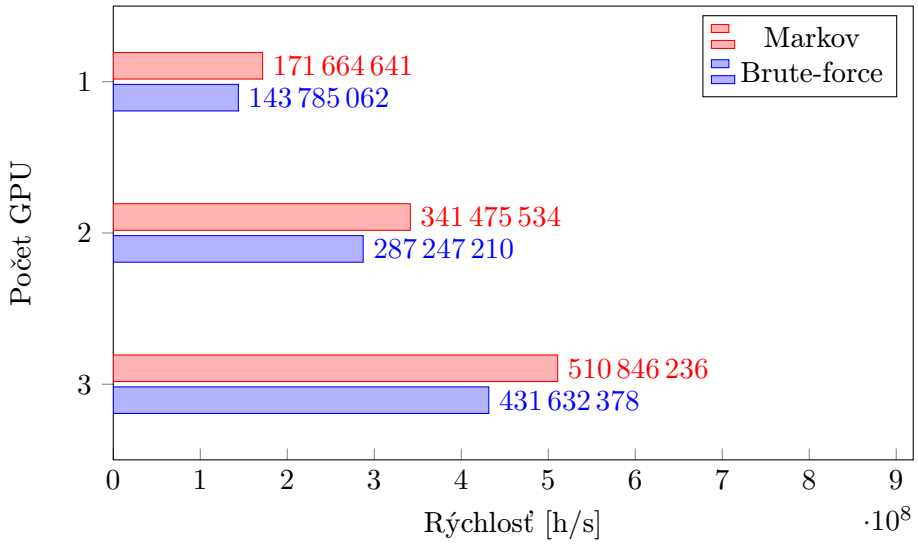
Rýchlosť určíme z celkovej doby, ktorá je potrebná pre nájdenie hesla. To vyžaduje stanovanie počtu hesiel, ktoré je potrebné vygenerovať k nájdeniu správneho hesla. Pri brute-force generátore je stanovanie tohoto počtu jednoduché, nakoľko presne poznáme proces obmeny znakov. V prípade generátora využívajúceho Markovský model bolo potrebné doimplementovať funkciu, ktorá zabezpečí vrátenie hesla zodpovedajúceho zadanému indexu. Heslá volíme s ohľadom na dostatočne dlhú dobu trvania (10-15 minút), aby bol minimalizovaný vplyv inicializácie.

7.2.3 Výsledky experimentu

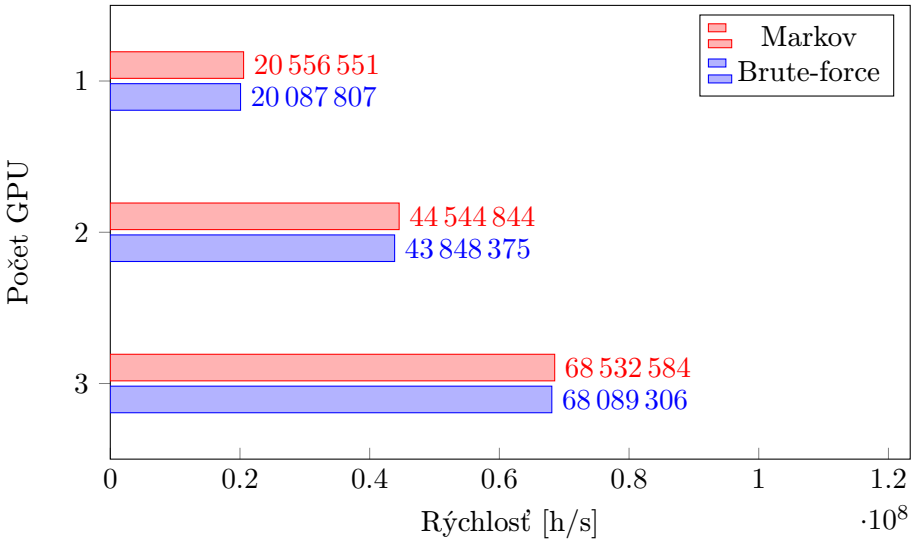
Grafy 7.2 a 7.3 zobrazujú namerané rýchlosti pri obnove hesiel pomocou GPU. Pri PDF formáte je na prvý pohľad zrejmé výrazné navýšenie rýchlosti v prípade použitia Markovského generátora. Toto navýšenie môže byť spôsobené najmä nižším objemom dát, ktoré sú počas generovania prenášané do GPU. Menej patrný rozdiel v rýchlostiach je pri formáte DOC, čo potvrdzuje našu hypotézu.

Výsledky meraní vykonávané na CPU je možné vidieť v grafoch 7.4 a 7.5. Markovský generátor je v obidvoch prípadoch pomalší, čo môže byť spôsobené najmä veľkým počtom prístupov do pamäte pri tvorbe každého hesla. Nakoľko je však prelamovanie na CPU využívané len pre krátke heslá, je tento rozdiel zanedbateľný.

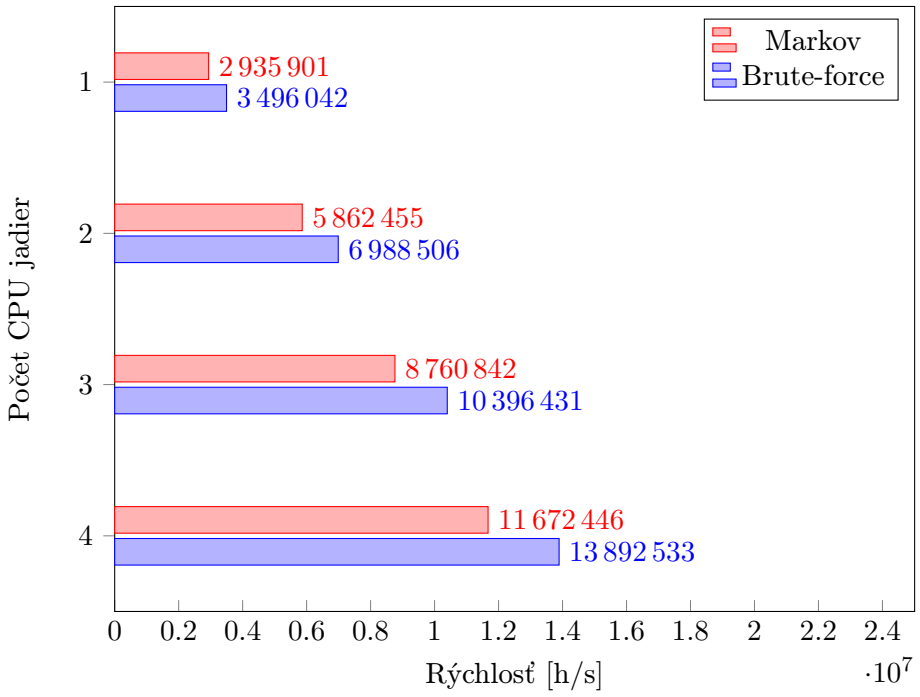
Vo všetkých meraniach je možné vidieť výbornú škálovateľnosť pôvodného aj súčasného riešenia. Navýšenie rýchlosti je v niektorých prípadoch takmer priamo úmerné počtu výpočetných prostriedkov.



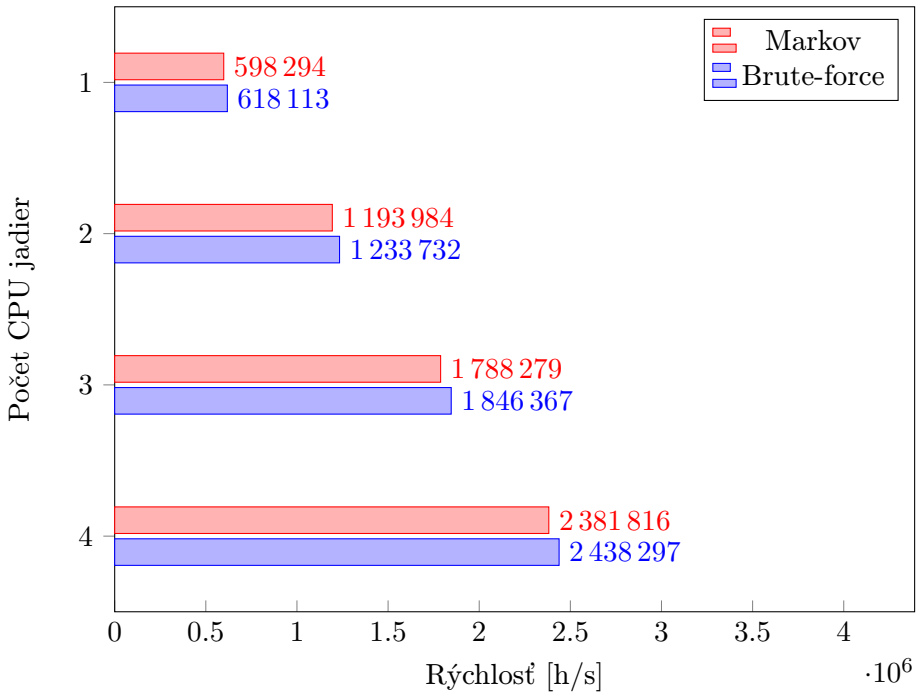
Obr. 7.2: Rýchlosť obnovy hesla PDF súboru s bezpečnostnou revíziou 5 na GPU



Obr. 7.3: Rýchlosť obnovy hesla DOC súboru vo verzii 97 na GPU



Obr. 7.4: Rýchlosť obnovy hesla PDF súboru s bezpečnostnou revíziou 5 na CPU



Obr. 7.5: Rýchlosť obnovy hesla DOC súboru vo verzii 97 na CPU

Kapitola 8

Záver

Dôsledkom narastajúcej dĺžky hesiel a využívaním nových šifrovacích algoritmov je z časového hľadiska nemožná obnova hesla pri použití útoku hrubou silou. Z tohoto dôvodu sa ukazuje nevyhnutné použitie sofistikovanejších metód pri obnove hesiel.

V tejto práci bol vykonaný rozbor používaných metód založených na frekvenčnej analýze, najmä Markovských modelov a jednoduchých regulárnych výrazov, ktoré umožňujú dosiahnutie tohoto cieľa. V prípade Markovských modelov sme analyzovali použitie Markovského modelu 1. rádu a vrstvového Markovského modelu. Nakoľko sa ukázalo, že prínos jednotlivých metód je závislý na druhu slovníka, na ktorom je vykonávaná frekvenčná analýza, rozhodli sme sa pre použitie obidvoch modelov. Na základe týchto informácii sme navrhli paralelný algoritmus pre generovanie hesiel, ktorý kombinuje všetky tieto prístupy.

Navrhnutý algoritmus bol implementovaný v prostredí OpenCL ako rozšírenie existujúceho nástroja Wrathion. Na záver bolo implementované riešenie porovnané s pôvodným útokom hrubou silou z hľadiska dosahovanej rýchlosti. Okrem toho bola experimentálne dokázaná prínosnosť Markovského modelu v procese obnovy hesiel.

Literatúra

[1] Florêncio, D.; Herley, C.; Oorschot, P.: An Administrator's Guide to Internet Password Research. In *Proceedings of the 28th USENIX Conference on Large Installation System Administration*, LISA'14, Berkeley, CA, USA: USENIX Association, 2014, ISBN 978-1-931971-17-1, s. 35–52.

[2] Hranický, R.; Matoušek, P.; Ryšavý, O.; et al.: Experimental Evaluation of Password Recovery in Encrypted Documents. In *Proceedings of ICISSP 2016*, SciTePress - Science and Technology Publications, 2016, ISBN 978-989-758-167-0, s. 299–306.

[3] Ma, J.; Yang, W.; Luo, M.; et al.: A Study of Probabilistic Password Models. In *Security and Privacy (SP), 2014 IEEE Symposium on*, The University of Texas at Austin, ACM, 2014.

[4] Marachal, S.: Advances in password cracking. *Journal in Computer Virology*, ročník 4, č. 1, 2008: s. 73–81, ISSN 1772-9890.

[5] Munshi, A.: The OpenCL Specification Version 1.2. Khronos Group, 2012. URL https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf

[6] Narayanan, A.; Schmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM conference on Computer and communications security - CCS '05*, The University of Texas at Austin, ACM, 2005.

[7] Rabiner, L.: A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, ročník 77, 1989, s. 257–286.

[8] Schmied, J.: GPU akcelerované prolamování šifer. Diplomová práca, FIT VUT v Brně, Brno, 2014.

[9] Tansey, W.: Improved Models for Password Guessing. Technická správa, University of Texas, 2011.

[10] Wiemer, F.; Zimmermann, R.: High-speed implementation of bcrypt password search using special-purpose hardware. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, 2014, ISSN 2325-6532, s. 1–6.

Dodatok A

Obsah CD

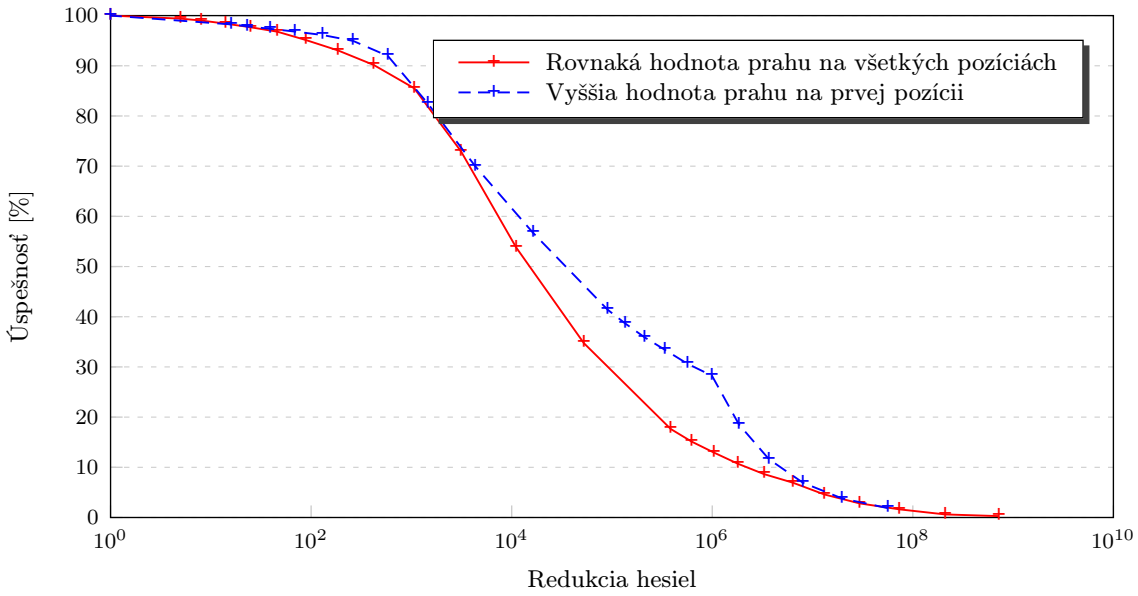
Na priloženom CD sa nachádza

- elektronická verzia tohoto dokumentu spolu zo zdrojovými súbormi v jazyku LaTeX,
- nástroj určený na vytváranie štatistík potrebných pre generovanie s využitím Markovského modelu,
- nástroj Wrathion rozšírený o generátor hesiel využívajúci Markovský model s podporou maskovania,
- nástroj pre experimentálne overenie úspešnosti pri generovaní hesiel s využitím Markovského modelu.

Obsah CD bližšie popisuje súbor `README.txt`, ktorý sa nachádza v koreňovom adresári.

Dodatok B

Kompletné výsledky experimentov



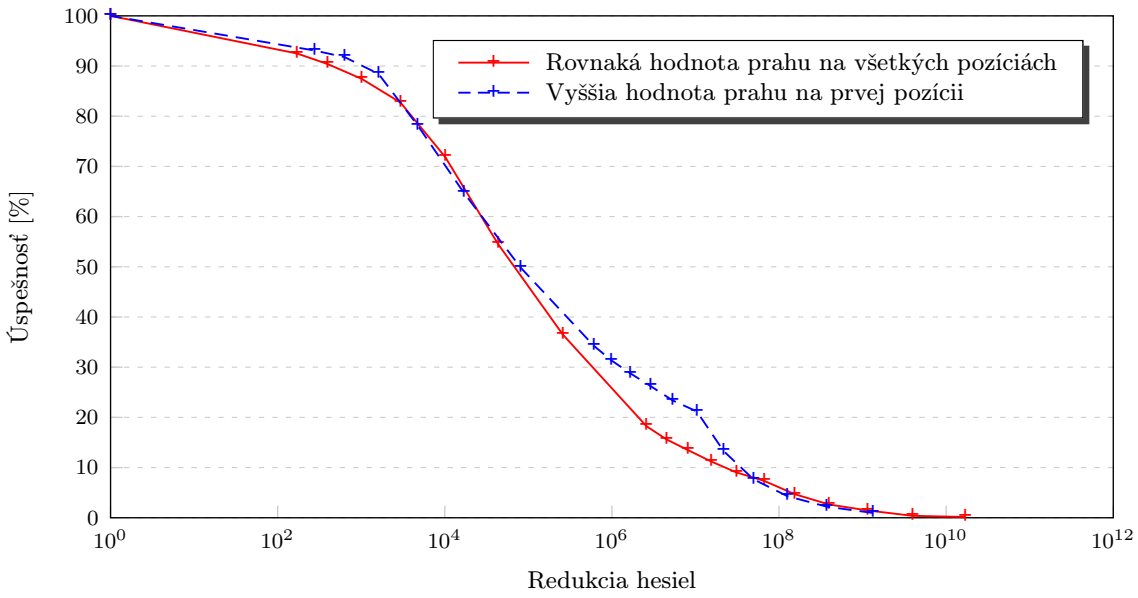
Obr. B.1: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 1–7 znakov

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	6 351 898	6,94	7,32	8,68
13	1 038 234	12,93	13,60	15,56
15	385 531	17,63	18,55	21,03
20	52 381	34,75	35,13	39,16
25	11 101	53,77	54,75	58,12
35	1 066	85,52	85,48	86,12
45	185	93,06	92,32	93,50
60	25	97,67	97,40	98,09
75	5	99,39	99,36	99,53

Tabuľka B.1: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 1–7 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,7	8 034 272	6,92	7,61	10,11
64,9	1 844 478	18,54	19,37	24,15
64,12	338 536	33,38	34,48	37,89
64,15	90 359	41,41	43,12	46,91
64,20	16 369	56,69	57,37	62,70
64,30	1 462	82,45	82,70	86,89
64,40	262	94,46	94,09	95,66
64,55	39	97,37	96,77	97,85
64,60	23	97,85	97,59	98,29

Tabuľka B.2: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 1–7 znakov.



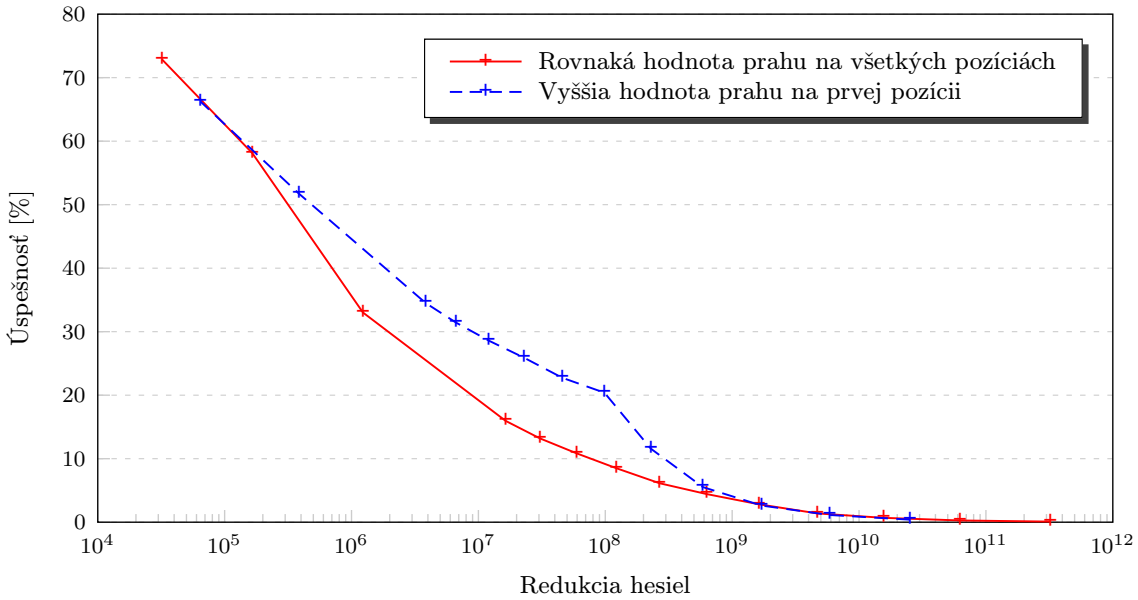
Obr. B.2: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 8 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	66 342 043	7,35	8,70	9,80
13	8 132 836	13,48	15,59	17,26
15	2 588 563	18,25	21,59	23,81
20	259 149	36,48	39,79	43,19
25	43 478	54,52	58,29	61,76
30	10 112	71,88	73,75	77,00
40	1 012	87,45	87,09	87,99
45	395	90,41	89,82	90,97
50	170	92,51	91,96	93,17

Tabuľka B.3: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 8 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,9	21 672 614	13,31	14,75	18,22
64,12	2 892 945	26,26	29,15	31,81
64,15	606 694	34,28	38,97	42,15
64,20	80 984	49,38	51,51	58,24
64,25	16 984	64,70	68,93	73,29
64,30	4 740	78,16	80,20	84,26
64,35	1 611	88,44	88,40	89,92
64,40	633	91,78	91,24	92,65
64,45	277	93,09	92,37	93,83

Tabuľka B.4: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 8 znakov.



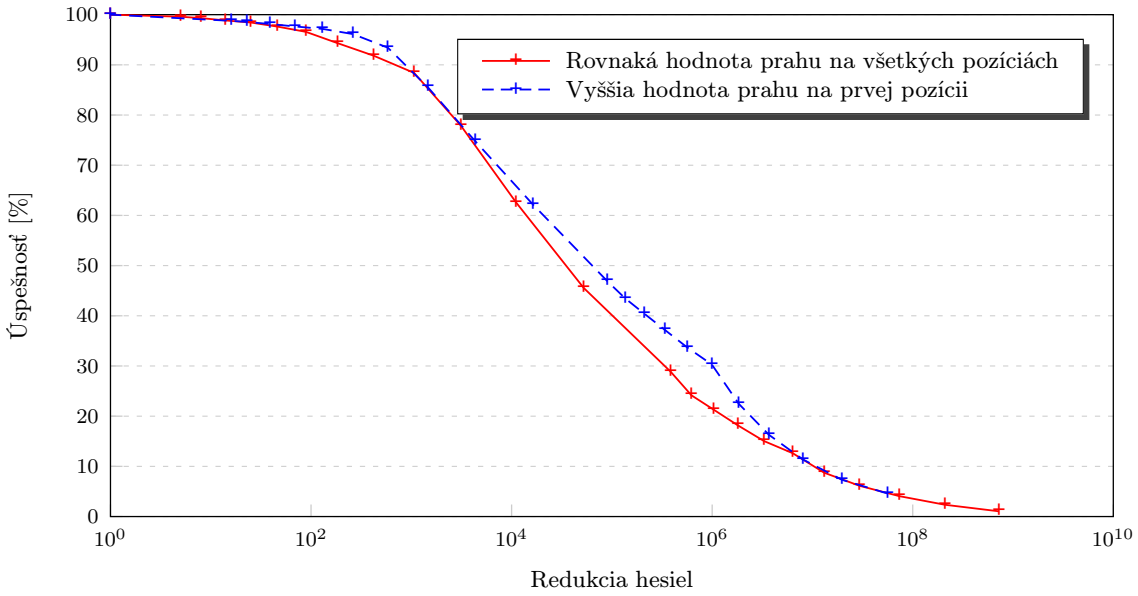
Obr. B.3: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 9 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	630 249 410	4,43	5,94	6,95
11	267 287 274	6,14	8,02	9,37
12	122 146 558	8,50	10,33	12,07
13	59 432 264	10,84	13,13	14,90
14	30 504 234	13,20	16,15	18,26
15	16 394 233	15,96	19,57	21,94
20	1 230 956	33,02	36,12	40,02
25	165 216	58,12	61,84	65,71
30	32 020	72,89	74,91	78,26

Tabuľka B.5: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 9 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,9	228 766 484	11,57	13,45	17,06
64,10	98 476 470	20,39	23,18	24,97
64,11	45 940 000	22,73	25,92	28,16
64,12	22 902 480	25,90	28,96	31,71
64,13	12 072 179	28,59	32,15	34,86
64,14	6 672 801	31,42	35,70	38,83
64,15	3 842 398	34,51	39,54	42,94
64,20	384 674	51,74	55,42	60,47
64,25	64 538	66,23	70,38	75,25

Tabuľka B.6: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Faithwriters, Hotmail, Myspace, PhpBB. Evaluácia: Rockyou s heslami s dĺžkou 9 znakov.



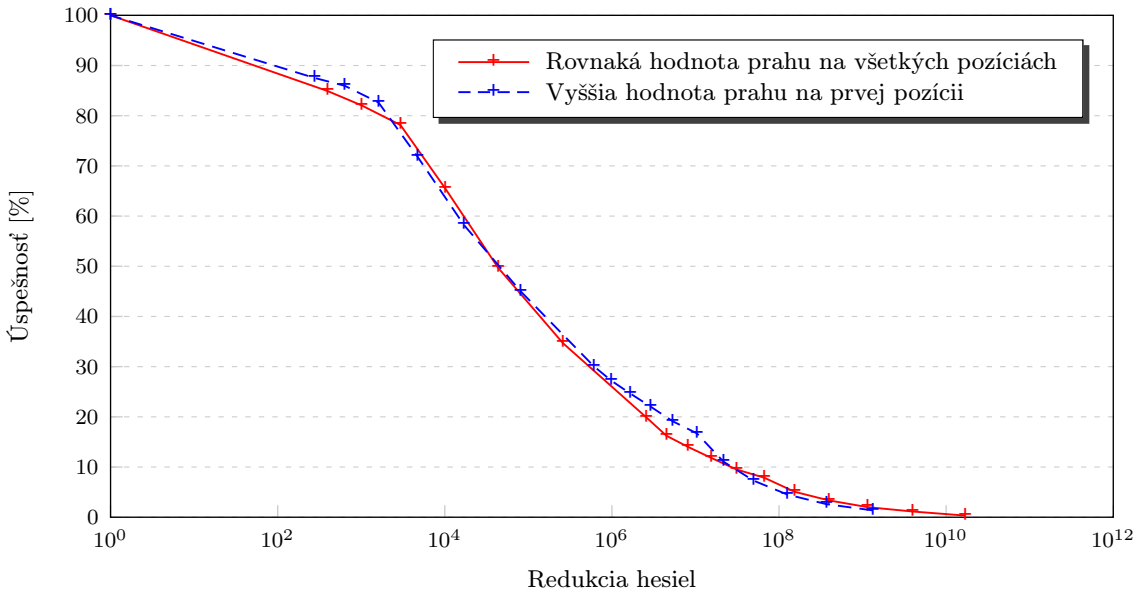
Obr. B.4: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 1–7 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	6 351 898	12,62	13,62	14,75
13	1 038 234	21,82	22,67	24,56
15	385 531	28,86	30,61	32,59
20	52 381	45,55	46,78	49,22
25	11 101	62,57	63,53	65,94
35	1 066	88,40	88,80	89,35
45	185	94,30	94,44	94,60
60	25	98,44	98,53	98,70
75	5	99,59	99,66	99,73

Tabuľka B.7: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 1–7 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,7	8 034 272	11,37	12,84	15,57
64,9	1 844 478	22,45	24,45	28,49
64,12	338 536	37,17	39,67	42,78
64,15	90 359	46,87	49,79	52,86
64,20	16 369	62,14	63,90	67,00
64,30	1 462	85,53	86,62	88,90
64,40	262	96,13	96,69	96,93
64,55	39	98,14	98,27	98,47
64,60	23	98,54	98,64	98,81

Tabuľka B.8: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 1–7 znakov.



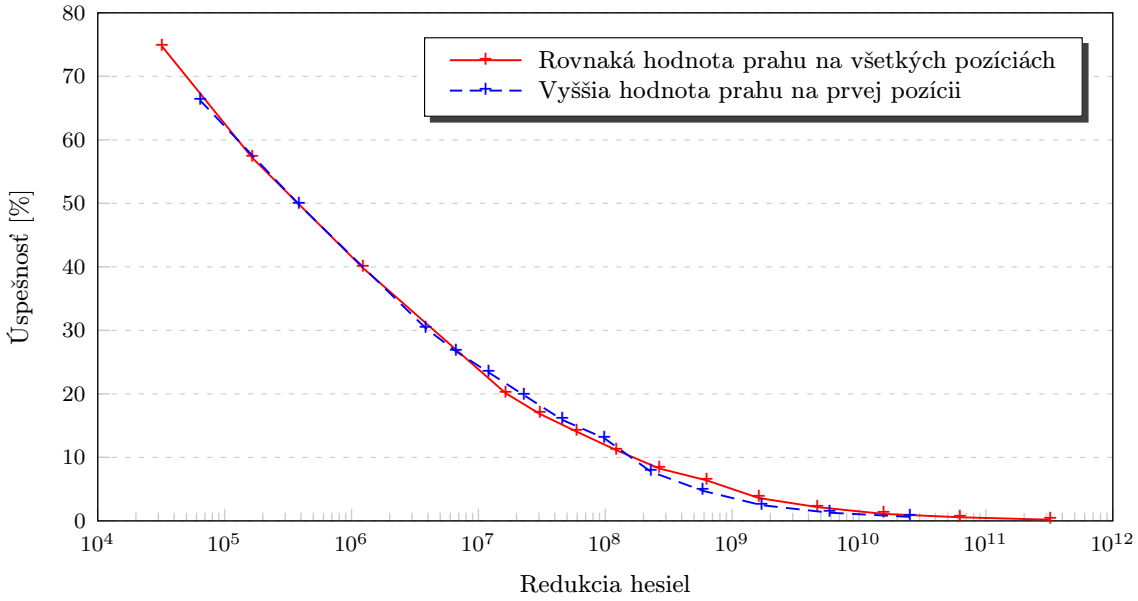
Obr. B.5: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 8 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	66 342 043	7,88	8,77	9,59
13	8 132 836	14,02	15,54	17,02
15	2 588 563	19,87	22,01	23,59
20	259 149	34,69	36,56	38,90
25	43 478	49,70	52,22	54,92
30	10 112	65,52	67,52	67,25
35	2 946	78,15	78,50	79,33
40	1 012	82,05	82,60	82,87
45	395	84,92	85,22	85,43

Tabuľka B.9: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 8 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,9	21 672 614	11,06	12,63	14,87
64,12	2 892 945	22,06	24,23	26,46
64,15	606 694	30,09	33,31	35,85
64,20	80 984	45,00	47,67	50,61
64,25	16 984	58,21	61,49	64,80
64,30	4 740	71,78	74,04	77,07
64,35	1 611	82,63	83,24	84,24
64,40	633	85,97	86,96	87,03
64,45	277	87,52	87,92	88,16

Tabuľka B.10: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 8 znakov.



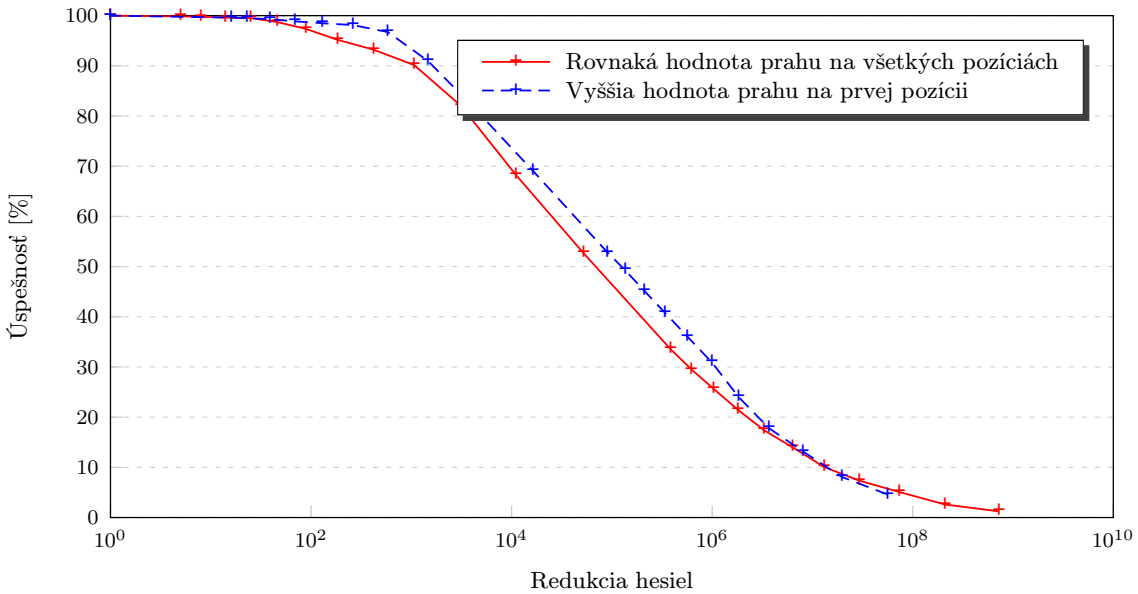
Obr. B.6: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 9 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	630 249 410	6,38	7,60	8,48
11	267 287 274	8,24	9,78	10,93
12	122 146 558	11,14	12,57	11,14
13	59 432 264	14,06	15,24	17,37
14	30 504 234	16,84	18,84	21,07
15	16 394 233	20,11	22,37	24,58
20	1 230 956	39,82	41,80	45,40
25	165 216	57,16	59,89	63,80
30	32 020	74,75	76,15	79,25

Tabuľka B.11: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 9 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,9	228 766 484	7,74	9,92	11,71
64,10	98 476 470	13,01	15,44	17,26
64,11	45 940 000	15,90	18,65	20,88
64,12	22 902 480	19,79	22,03	24,82
64,13	12 072 179	23,37	25,39	28,88
64,14	6 672 801	26,68	29,64	33,09
64,15	3 842 398	30,35	33,84	37,27
64,20	384 674	49,82	52,81	57,21
64,25	64 538	66,16	69,68	74,32

Tabuľka B.12: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Rockyou. Evaluácia: PhpBB s heslami s dĺžkou 9 znakov.



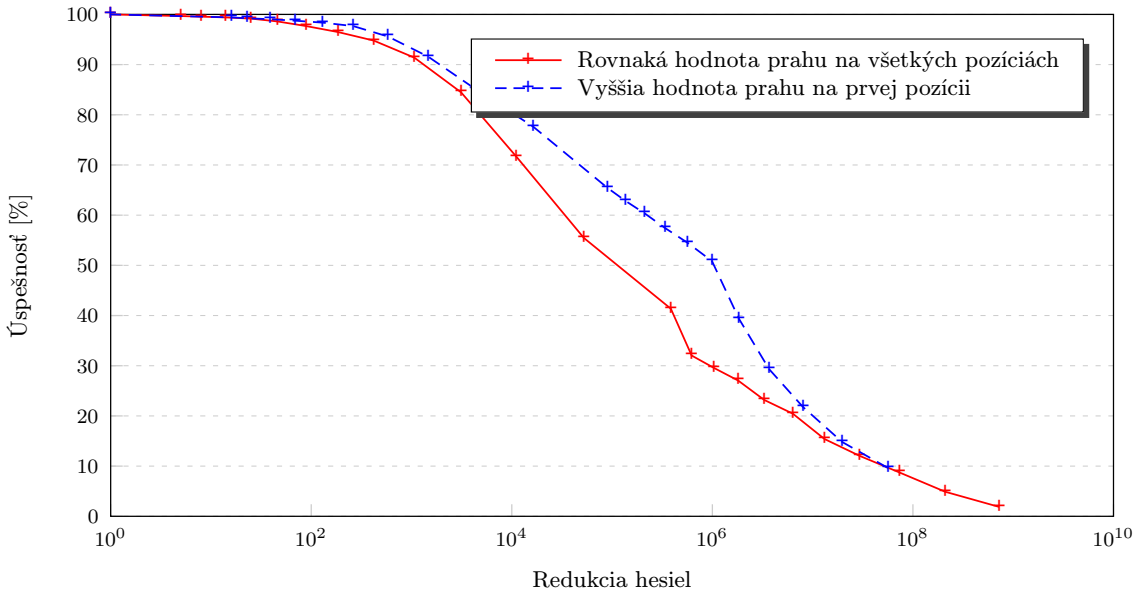
Obr. B.7: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Rockyou. Evaluácia: Faithwriters s heslami s dĺžkou 1–7 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	6 351 898	13,98	15,86	17,13
13	1 038 234	25,61	27,70	29,67
15	385 531	33,53	35,96	37,94
20	52 381	52,64	54,72	56,84
25	11 101	68,21	69,61	71,44
35	1 066	90,16	90,34	90,55
45	185	95,17	95,54	95,59
60	25	99,51	99,63	99,67
75	5	99,86	99,95	99,95

Tabuľka B.13: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Rockyou. Evaluácia: Faithwriters s heslami s dĺžkou 1–7 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,7	8 034 272	13,16	15,42	18,57
64,9	1 844 478	23,96	27,37	31,55
64,12	338 536	40,79	45,14	48,46
64,15	90 359	52,82	57,35	60,20
64,20	16 369	69,05	71,93	74,62
64,30	1 462	90,99	91,87	93,68
64,40	262	98,12	98,42	98,54
64,55	39	99,28	99,44	99,49
64,60	23	99,51	99,63	99,67

Tabuľka B.14: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Rockyou. Evaluácia: Faithwriters s heslami s dĺžkou 1–7 znakov.



Obr. B.8: Percentuálna úspešnosť v závislosti na redukcii hesiel pri použití klasického Markovského modelu. Trénovenie: Rockyou. Evaluácia: Hotmail s heslami s dĺžkou 1–7 znakov.

Prah	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
10	6 351 898	20,44	21,04	22,56
13	1 038 234	29,56	30,18	32,27
15	385 531	41,36	42,82	44,46
20	52 381	55,45	56,62	58,49
25	11 101	71,66	73,27	74,23
35	1 066	91,39	91,84	92,10
45	185	96,48	96,45	96,60
60	25	99,17	99,28	99,34
75	5	99,67	99,73	99,76

Tabuľka B.15: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití rovnakej hodnoty prahu pre všetky pozície hesla. Trénovenie: Rockyou. Evaluácia: Hotmail s heslami s dĺžkou 1–7 znakov.

Prah [prvá pozícia, ostatné pozície]	Redukcia	Úspešnosť [%]		
		Klasický	Vrstvový	Kombinácia
64,7	8 034 272	21,75	21,04	26,85
64,9	1 844 478	39,33	40,14	47,17
64,12	338 536	57,42	59,03	62,51
64,15	90 359	65,41	67,76	70,29
64,20	16 369	77,62	78,96	81,50
64,30	1 462	91,60	92,28	93,56
64,40	262	97,68	98,00	98,09
64,55	39	99,05	99,17	99,28
64,60	23	99,23	99,34	99,43

Tabuľka B.16: Úspešnosti dosiahnuté klasickým a vrstvovým Markovským modelom a ich kombináciou pri použití prahu s hodnotou 64 na prvej pozícii hesla. Trénovenie: Rockyou. Evaluácia: Hotmail s heslami s dĺžkou 1–7 znakov.