Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Metódy útoku hrubou silou na TrueCrypt

Diplomová Práca

2016 **Martin Strapko**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Metódy útoku hrubou silou na TrueCrypt

Diplomová Práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra Informatiky
Vedúci práce: RNDr. Richard Ostertág PhD.

2016 **Martin Strapko**

Poďakovanie

Abstrakt

Cieľom práce je napísať program, ktorý sa snaží hrubou silou, ale čo najefektívnejšie, nájsť heslo pre dešifrovanie disku zašifrovaného pomocou programu TrueCrypt.

KĽÚČOVÉ SLOVÁ: TrueCrypt, útoky hrubou silou, bezkontextové gramatiky

Abstract

The main purpose of this thesis is to implement application that is able effectively recover password for encrypted disk with TrueCrypt

KEYWORDS: TrueCrypt, brute-force attacks, context-free grammars

Obsah

Zoznam listingov

Zoznam obrázkov

Úvod

V dnešnom svete fungujúcom na elektronických dátach, ktoré pre nás majú obrovskú cenu, sa ľudia snažia udržať ich čo najviac v tajnosti. Za týmto účelom vznikli mnohé programy slúžiace na zašifrovanie dát pomocou používateľského hesla. Vzhľadom na rýchly vzrast výpočtovej sily počítačov sa skrátil čas potrebný na prehľadanie celého priestoru možných hesiel. Preto sa používateľské heslá začali predĺžovať a komplikovať.

Zatiaľ čo vďaka tomuto trendu sa zvýšila bezpečnosť dát, heslá sa stali na toľko komplikované, že boli ťažko zapamätateľné. Zatiaľ čo spoločnosti poskytujúce služby využívajúce zaheslované dáta dokážu vyresetovať používateľovi jeho heslo aby mal možnosť zvoliť si nové, zašifrovaný disk na domácom počítači takúto možnosť nemá. Práve tento problém sa snažíme adresovať v tejto práci. Implementujeme program, ktorý by na základe pomocnej informácie pri vstupe čo najrýchlejšie našiel používateľové stratené heslo.

Takéto hľadanie heslá sa v prípade útokov hrubou silou skladá z dvoch fáz. Prvou fázou je generovanie hesiel, ktoré pri tomto útoku budeme skúšať použiť. Práve na túto fázu sa budeme sústrediť v tejto práci. Budeme hľadať spôsob akým čo najefektívnejšie generovať tieto heslá. Druhou fázou je samotné overenie správnosti zadaného hesla. Túto časť nebudeme priamo implementovať, avšak výstup nášho programu by mal byť použiteľný s ľubovolným voľne dostupným programom na skúšanie hesiel ako napríklad *hashCat*.

KAPITOLA 1

TrueCrypt

TrueCrypt je šifrovací program poskytujúci používateľovi možnosť zašifrovať ľubovoľnú časť disku v počítači pomocou používateľom zvoleného hesla. Vývoj tohto programu bol ukončený roku 2014 a podľa autorov nie je bezpečný, nakoľko jeho implementácia môže obsahovať bezpečnostné chyby. Cieľom tejto práce nie je odhalenie týchto chýb, nakoľko sa nesnažíme zlomiť toto šifrovanie. Našim cieľom je implementácia algoritmu schopného zistiť stratené používateľské heslo za účelom odšifrovania dát ich majiteľom. Nižšie bližšie popíšeme algoritmus šifrovania disku pomocou programu TrueCrypt a v následnovných kapitolách podrobne rozoberieme možnosti útokov na používateľské heslá.

Ako sme spomínali v úvode program TrueCrypt slúži na zašifrovanie dát na používateľskom disku pomocou zvoleného hesla. Tento program má implementované viaceré šifrovacie algoritmy, avšak predpokladáme, že poznáme algoritmus zvolený na zašifrovanie cieľového disku. Zašifrovanie disku pomocou tohto programu prebieha v dvoch fázach. Najskôr vygeneruje náhodný kľúč a pomocou neho zašifruje disk aj s dátami. Následne zapíše konfiguráciu a vygenerované kľúče do hlavičky zašifrovaného disku a tú následne zašifruje kľúčom vygenerovaným pomocou používateľského hesla. Dôležitou informáciou o programe TrueCrypt je že zdrojové kódy tohto programu sú voľne dostupné na internete.

KAPITOLA 2

Útoky na šifrovanie

Existuje viacero spôsobov akými sa dá získať zašifrované dáta bez znalosti kľúča, ktorým sú zabezpečené. Jedným z týchto spôsobov je nazvaný útok hrubou silou. Podstatou tohto útoku je skúšanie všetkých možností. Samozrejme existuje viacero metód implementácie tohto útoku, ktoré si popíšeme v následujúcej kapitole.

2.1 Inkrementálna metóda

Inkrementálna metóda je pravdepodobne najintuitívnejší útok hrubou silou. Pri tomto type útoku útočník generuje všetky reťazce vstupnej abecedy kratšie ako stanovená maximálna dĺžka hľadaného hesla. Ak poznáme túto maximálnu dĺžku hľadaného hesla, tak tento prístup ma 100 percentnú úspešnosť keďže prehľadá celý priestor možných hesiel. Avšak týchto hesiel je <strasne vela, treba dosadit vzorec>, čo by pri skúšaní <pekna konstanta> hesiel za sekundu trvalo dokopy <ohurujuce cislo> rokov.

2.2 Slovníkový útok

Iná možnosť ako skúšať všetky možné kombinácie znakov je skúšať heslá, ktoré sú často používane alebo majú nejaký konkrétny význam pre používateľa. Z týchto slov vystaviame slovník pomocou ktorého následne skúšame či sa nám podarilo nájsť správne heslo. Keďže slovník použitý pri útoku sami pripravíme, vyskúšanie všetkych hesiel, ktoré obsahuje, bude uskutočniteľné v rozumne krátkom čase. Vďaka tomuto patrí medzi najpopulárnejšie metódy útoku. Avšak úspešnosť tohto útoku je závislá od hesiel, ktoré pridáme do slovníku. V tomto probléme nám môže pomôcť fakt, že hľadáme stratené heslo, čiže používateľ dokáže poskytnúť veľké množstvo potenciálnych hesiel, ktoré majú vysokú pravdepodobnosť úspechu.

2.2.1 Prekrúcanie slov

Avšak samotný používateľ nemusí vedieť hľadané heslo, ktoré by sa mohlo líšiť od niektorého v slovníku len v jednom znaku, napríklad vo veľkom alebo malom písmene. Preto sa často so slovníkovým útokom používa takzvané prekrúcanie slov, ktoré pomocou predom definovaných pravidiel skúsi znetvoriť postupne každé heslo zo slovníka. Týmto výrazne zväčší veľkosť slovníka čím zvýši šancu na úspech a zmenší sa množstvo hesiel, ktoré treba ručne generovať.

2.3 Hybridný útok

Metóda, ktorou sa zaoberáme v tejto práci a ktorú sme implementovali je hybrid vytvorený zo všetkých vyššie uvedených. Našim hlavným cieľom je nájsť hľadané heslo v konečnom čase. Preto sa budeme snažiť vytvoriť algoritmus, ktorý vygeneruje všetky možné reťazce kratšie ako zadaná maximálna dĺžka. Avšak aby sme tento čas minimalizovali, použijeme slovník obsahujúci často používané heslá a heslá o ktorých si používateľ myslí, že by mohli byť správne. Naštudujeme si štruktúru hesiel, ktoré používateľ používa. Z týchto znalostí si vytvoríme pravidlá pomocou ktorých budeme generovať naše finálne pokusy na odšifrovanie cieľového disku.

KAPITOLA 3

Učenie

Ako sme spomínali vyššie v texte, náš program bude generovať heslá na základe nejakých znalostí. Tento proces učenia sa, sa dá implementovať pomocou viacerých známych metodík, medzi ktoré patria napríklad Markovové zdroje alebo pravdepodobnostné gramatiky.

3.1 Pravdepodobnostné bezkontextové gramatiky

Bezkontextové gramatiky používajú pravidlá pri ktorých sa neterminál môže zmeniť na ľubovoľnú vetnú formu bez ohľadu na kontext v ktorom sa nachádza. Pravdepodobnostné gramatiky vzniknú keď každému pravidlu priradíme číslo v rozmedzí 0 a 1. Toto číslo vyjadruje pravdepodobnosť použitia daného pravidla pre jeho neterminál, súčet pravdepodobností jedného neterminálu by mal byť rovný 1. Následne pravdepodobnosť terminálnej vetnej formy je rovná súčinu pravdepodobností pravidiel použitých v jej odvodení. V gramatikách je možné a častokrát až bežné aby jedna terminálna vetná forma mala viacero stromov odvodenia, poradí použitia pravidiel. My sme sa tejto vlastnosti snažili vyhnúť, pretože by sme zbytočne generovali jedno heslo viac krát.

3.2 Markovové zdroje

Náhodný proces prechadzajúci cez priestor stavov sa nazýva Markovov zdroj ak spĺňa Markovovu vlastnosť. Táto vlastnosť je popísaná ako takzvaná bezpamäťovosť a hovorí o tom, že distribúcia pravdepodobností následujúceho stavu závisí len od terajšieho stavu a nezáleži na sekvencií udalostí, ktoré mu predchádzali. Práve táto vlastnosť nás doviedla k skúmaniu tohto riešenia. Hlavným problémom použiteľností bezkontextových gramatík bolo množstvo pamäte, ktoré vyžadovali, jak pri vytváraní gramatiky tak aj pri generovaní hesiel. Markovové zdroje ako jedinú informáciu zo vstupného slovníka si zachovávajú pravdepodobnosti výskytu znakov po predom definovanom prefixe.

V prípade, že algoritmus narazí na neznámy prefix, predpokladá že všetky znaky majú rovnakú pravdepodobnosť. Medzera a znak nového riadku sa tiež počítajú medzi tieto znaky, a vďaka ním sa naučí algoritmus vytvárať reťazce podobnej dĺžky ako tie čo dostal na vstupe.

KAPITOLA 4

Implementácia

4.1 Bezkontextová gramatika

4.1.1 Tvorba gramatiky

Pri tvorbe gramatiky sme potrebovali zaistiť aby gramatika spĺňala určité podmienky. Prvou z nich je schopnosť gramatiky vygenerovať všetky reťazce zo vstupnej abecedy kratšie ako používateľom zadaná maximálna dĺžka. Druhou podmienkou je aby každé terminálne slovo, ktoré gramatika generuje malo práve jeden strom odvodenia. Nakoniec by sme chceli aby algoritmus, ktorý bude pomocou tejto gramatiky generovať heslá bol deterministický.

Jednodché neterminály Prvý typ neterminálov, ktoré budeme nazývať jednoduché, obsahuje pravidlá na zterminalnenie generovaného slova. Keďže naša vstupná abeceda obsahuje okolo 70 znakov, medzi ne patria veľke a malé písmena, cifry a niektoré často používané symboly, rozhodli sme sa ich rozdeliť do jednotlivých skupín. Pre každú z týchto skupín sme vytvorili neterminál, ktorý bude reprezentovať sekvenciu pevnej dĺžky zloženú zo znakov danej skupiny. V gramatike tieto neterminály vyjadrujeme pomocou prvého písmena anglického názvu danej skupiny.

- U - upper case - veľké písmena
- L - lower case - malé písmena
- D - digit - cifry
- S - symbol - symboly

Každý jednoduchý neterminál sa teda skladá z písmena vyjadrujúceho skupinu znakov, ktoré generuje, a čísla popisujúceho dĺžku sekvencie na pravej strane pravidiel tohto neterminálu. Keďže generovanie všetkých variacií veľkostí k pri n prvkoch môže byť

obrovské množstvo rozhodli sme sa zadefinovať maximálnu dĺžku sekvencie generovanej jednoduchým neterminálom.

Zložené neterminály Jednoduché neterminály nám pomáhajú vyjadrovať sekvenciu znakov práve jedného z vyššie vymenovaných typov. Aby sme boli schopný popísať ľubovolný reťazec tvorený znakmi vstupnej abecedy, budeme tieto jednoduché neterminály skladať do skupín, zložených neterminálov. Tieto neterminály vyjadrujú vzdy jeden možný predpis pre terminálne slovo. Napríklad neterminál *U1L3D4* vyjadruje všetky terminálne slová začínajúce na veľké písmeno nasledované tromi malými písmena, ukončené štvoricou cifier.

Ďalej taktiež nedovoľujeme aby sa vyskytovali 2 jednoduché neterminály rovnakého typu za sebou. V prípade, že potrebujeme popísať sekvenciu terminálnych znakov jedného typu dlhšiu ako povolené maximum (popísane vyššie), rozdelíme túto sekvenciu do viacerých jednoduchých neterminálov pažravým algoritmom, čiže každý z týchto neterminálov zobere maximálný možný počet znakov sekvencie. Tento spôsob nám zaručí, že nevzniknú dva rôzne zložené neterminály vyjadrujúce ten istý predpis terminálneho slova.

Počiatočný neterminál gramatiky Z bude obsahovať pravidlá prepisujúce tento neterminál na niektorý zo zložených alebo jednoduchých neterminálov. Tento spôsob generovania gramatiky spĺňa obe pravidlá, ktoré sme popisovali v úvode tejto kapitoly.

4.1.2 Počítanie pravdepodobností

Ako sme spomínali v úvode textu, nami generované pokusy o nájdenie hesla chceme prispôsobit potrebám jednotlivých používateľom, ktorí sa snažia získať svoje stratené heslo. Aby sme vedeli čo najlepsie vyhovieť týmto používateľom, potrebujeme upraviť našu gramatiku. Tu prichádzajú do pozornosti pravdepodobností jednotlivých pravidiel našej gramatiky. Našim cieľom je nastaviť našu gramatiku tak aby generovala heslá podľa pravdepodobnosti použitia daným používateľom. Úspešnosť tohto učenia gramatiky bude drastický záležať od kvality vstupných dát.

Vzhľadom na to, že v dnešnom svete používatelia používajú rôzne služby, ktoré každá odporúča mať jedinečné heslo, používatelia používajú niekoľko hesiel naraz. Tieto heslá by si radi všetky pamätali a preto si často vytvoria pre seba charakteristický spôsob tvorby a zapamätania si týchto hesiel. V ideálnom prípade by sme chceli aby naše vstupné dáta pozostávali z čo najväčšieho počtu hesiel vytvorených pomocou tohto charakteristického spôsobu, keďže každé upresnenie informácií o hľadanom hesle nám zvýši rýchlosť nájdenia tohto hesla.

Keďže cieľom našej práce je nájsť heslo so 100% pravdepodobnosťou, čo v najhoršom prípade znamená vygenerovať všetky možné reťazce kratšie ako zadaná maximálna dĺžka hesla, tak pravidla našej gramatiky sa budú meniť len pri zmene maximálnej dĺžky hesla. V ostatných prípadoch sa budú meniť len ich pravdepodobností. Pravdepodobností terminálných sekvencií budeme rátať ako percento výskytov danej terminálnej sekvencie spomedzi všetkých sekvencií spadajúcich pod tento neterminál. Práve kvôli tomuto spôsobu sme pridali v implementácií možnosť napísať do vstupného slovníku počty výskytov jednotlivých hesiel, aby mal používateľ možnosť zdôrazniť dôležitosť hesla. Pri počítaní pravdepodobností zložených neterminálov máme viacero možností ako postupovať.

Priamo zo vstupného slovníka Prvý spôsob ako postupovať bol identický s tým pre jednoduché neterminály. Pre každe pravidlo gramatiky prepisujúce neterminál Z na zvolený zložený neterminál vypočítame jeho pravdepodobnosť ako pomer počtu výskytov tohto neterminálu a všetkých výskytov. Tento spôsob môže mať ešte 2 varianty.

- Do výskytov počítame len výskyty hesiel ktoré sú presne reprezentované daným neterminálom
- Do výskytov započítame aj výskyty kedy je zvolený neterminál podreťazcom iného neterminálu

Rekurzívne Ďalší spôsob spočíva v tom, že zo vstupného slovníka vypočítame pravdepodobností len pre jednoduché netermiály. Následne pre zložené neterminály počítame pravdepodobnosti ako súčin pravedpodobností jednoduchých neterminálov, ktoré daný neterminál obsahuje.

Všetky vyššie spomenuté metódy na počítanie pravdepodobností pravidiel gramatiky sme implementovali. Ich vzájomne porovnanie ako aj porovnanie s inými bežne používanými programami je vidieť v kapitole Výsledky. Bohužial sme zistili, že gramatika vytvorená naším algoritmom zaberala príliš veľa miesta na disku. Rozhodli sme sa skúsiť upraviť našu gramatiku tak aby vyžadovala menej zapamútaných pravidiel, zatiaľ čo by si udržala ostatné svoje vlastnosti.

4.1.3 Generovanie hesiel

Dôležitým aspektom používania bezkontextových gramatík je práve spôsob generovania hesiel. Našim hlavným cieľom bolo generovanie hesiel pomocou gramatiky od najpravdepodobnejšieho z nich. Tieto heslá generujeme tak, že počiatočný neterminál rozpíšeme na najpravdepodobnejší zložený neterminál. Následne jednoduché neterminály, z ktorých sa tento zložený neterminál skladá, prepíšeme postupne ich najpravdepodonejšími terminálnymi vetnými formami. K tomu sme potrebovali utriediť všetky

pravidlá pre jednotlivé neterminály zostupne podľa ich pravdepodobnosti. Toto utriedenie nám umožnilo pamätať si len indexy posledne použitých pravidiel jednotlivých neterminálov, ktoré práve rozpisujeme. Týmto spôsobom dokážeme popísať stromy odvodenia jednotlivých hesiel ako k-tice čísel vyjadrujúce poradie použitých pravidiel vrámci ich neterminálov. Kde jedno číslo slúži na určenie vybratého zloženého neterminálu a zvyšné vyjadrujú poradia použitých pravidiel *k-1* jednoduchých neterminálov, z ktorých sa tento zložený neterminál skladá.

Na začiatku je heslo s najvyššiou pravdepodobnosťou popísané vektorom samých núl. Z tohto bodu rozbehneme algoritmus prehľadávania do šírky s použitím prioritnej fronty. Ako prvé si do fronty pridáme všetky možné vektory indexov vzdialené od aktualného práve o 1, čiže také kde sa niektorý z indexov zvýši o jedna zatiaľ čo ostatné ostanú nezmenené. Do fronty pridávame dvojice vektor a pravdepodobnosť tohto vektoru. Pravdepodobnosť jednotlivých vektorov rátame ako súčin pravdepodobností pravidiel na ktoré ukazujú. Keďže každý čo pridávame sa líši práve v jednom indexe, p[t+1] = p[t] / pi[t] * pi[t+1]. Keď už sme pridali všetky takéto susedné vektory, vybereme z fronty ten s najvyššiou pravdepodobnosťou a na ňom celý proces opäť zopakujeme.

Týmto spôsobom by sme ale generovali veľké množstvo rovnakých vektorov, ktoré by sme dostali zmenou indexov v inom podarí. Napríklad ak by sme zvýšili najprv index na pozícii 1 a potom 3, dostali by sme to isté ako keby sme zvýšili na pozícii 3 a potom 1. Preto zavedieme ešte špeciálne číslo, ktoré názveme radom vektoru. Rád vektoru bude číslo určujúce pozíciu najvyššieho zmeneného indexu. Zároveň pri generovaní susedných vektorov dovolíme meniť indexy len na pozíciach výšších alebo rovných ako je aktuálny rád vektora. Týmto zariadíme aby sa negenerovali duplikáty, ktoré by sa líšili len v poradí akom boli indexy menené.

Tento priamočiarý prístup ku generovaniu spĺňa všetky naše požiadavky na generované heslá. Veľkosť fronty sa môže veľmi radikálne zmeniť na základe rozpoloženia pravdepodobností vrámci neterminálov. Preto by toto miesto bolo vhodné na použitie nejakej heuritisky. Bohužiaľ vrámci tejto práce sa nám nepodarilo nájsť vhodné heuristiky, ktoré by zmenšili pamäťovú náročnosť zatiaľ čo by čo najlepšie uchovali poradie hesiel.

4.2 Markovov zdroj

Po implementácií vyššie uvedeného algoritmu na generovanie hesiel pomocou pravdepodobnostných bezkontextových gramatík a odhalení nedostatkov čo sa týka pamäťovej náročnosti sme sa rozhodli implementovať ešte jednu metódu. Tou je Markovov zdroj.

Ako sme písali v predošlej kapitole, jedná sa o náhodný proces, ktorý spĺňa podmienku bezpamäťovosti. Markovove zdroje sa veľmi často používaju práve pri generovaní prirodzeného jazyka. Práve preto boli vhodný kandidát pre generovanie hesiel na základe znalostí získaných zo vstupného slovníka.

Bohužial táto metóda nespĺňa ani jednu z podmienok, ktoré sme si na začiatku definovali. Ako bolo písane jedná sa o náhodny proces, čiže dve od seba rôzne spustenia môžu viesť k rôznym výsledkom. Druhá podmienka o generovaní duplikátov taktiez nie je splnená, keďže tomuto zdroje nič nebráni k tomu vygenerovať viac krát počas behu to isté slovo a nič by mu v tom nemalo ani brániť, to je celá pointa bezpamäťovosti. A na koniec markovové zdroje nemusia generovať všetký možné reťazce kratšie ako zadaná maximálna dĺžka.

Aj keď prvé dve podmienky nevedia Markovove zdroje splniť už priamo z definície, s tretiou sme sa pokúsili niečo vymyslieť. Ako prvé sme sa pokúšali inicializovať všetky počty výskyt na 1 namiesto 0. Toto avšak spôsobilo, že sa zdroj relatívne ľahko dostal medzi prefixi, ktoré neboli definované, kde všetky znaky majú rovnakú pravdepodobnosť. Dôsledkom tohto bolo zacyklenie sa v týchto neznámych stavoch, čoho výsledkom boli dlhé nezmyselné reťazce znakov. Preto sme sa snažili nájsť spôsob ako nastaviť pravdepodobností nevidených stavov na nenulové, avšak dostatočne malé aby sa v nich samotný algoritmus nezacyklil.

KAPITOLA 5

Testy

Cieľom tejto kapitoly je podrobne popisať testy, ktoré slúžia k vyhodnoteniu efektivity a správanosti nášho algoritmu voči iným bežne dostupným riešeniam.

5.1 Časové testy

V teste budeme rátať čas behu jednotlivých programov od začiatku spracovania vstupného slovníka, až po vygenerovanie predom stanoveného počtu hesiel. Pri jednorázovom generovaní určitého počtu hesiel by naše riešenie mohlo byť trochu pomalšie ako iné bežne používané riešenie. Avšak v prípade, že budeme generovať heslá s rovankými parametrami rozdelené do viacerých osobitných požiadaviek, náš program by mohol byť trochu rýchlejší vďaka tomu, že nebude musieť spracovávať opäť vstupné dáta.

Vo viacerých spusteniach tohto testu sme menili premenné *maximálna dĺžka hesla*, *počet generovaných hesiel*, *počet opakovaných generovaní hesiel*, *maximálna dĺžka neterminálu v gramatike*. Taktiež budeme každý test spúšťať viac krát a vo výsledkoch uvedieme aritmetický priemer týchto hodnôt. Výsledne tabuľky a grafy sa nachádzajú v sekcií výsledky.

5.2 Test správnosti

Hlavným cieľom našej práce je najsť zabudnuté používateľské heslo v čo najkratšom čase. Toto sa dá dosiahnuť dvoma spôsobmi. Jeden z nich je optimalizácia kódu zodpovedného za overovanie či sme našli správne heslo. Druhý, ktorému sme sa v tejto práci venovali, je optimalizácia poradia generovania hesiel, ktoré chceme vyskúšať. Práve preto nám záleži aby nami sme minimalizovali počet hesiel, ktoré musíme vyskúšať než nájdeme to správne. Avšak toto môže veľmi záležať od prípadu k prípadu, preto bu-

deme náš program porovnávať s ostatnými bežne dostupnými riešeniami a ich spôsobmi generovania hesiel.

Vrámci testu necháme všetky programy vygenerovať predom stanovené množstvo hesiel zo zadaného slovníka. Tento slovník bude pozostávať z hesiel zoradených podľa počtu použití a bude identický pre všetky testované programy. Následne výsledne zoznamy hesiel porovnáme oproti pôvodnému slovníku. Ako jednu z hlavných metrík pri rozhodovaní o kvalite zoznamu hesiel budeme brať *štandardnú odchýlku* indexov hesiel oproti pôvodnému slovníku. Taktiež budeme sledovať iné miery kvality «TODO».

KAPITOLA 6

Výsledky

V tejto časti budú prezentované výsledky z testov popísaných v kapitole Testy. Ku každému testu bude tabuľka, graf a krátky popis vysvetľuúci čo sa z daných dát dá pochopiť. Zatiaľ bohužial kvôli problémom, ktoré sa naskytli pri spúšťaní testov je to tu prázdne

Literatúra