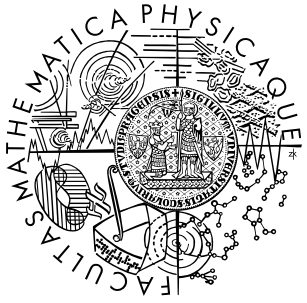
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁRSKA PRÁCA



Ivor Kollár

Steganografický souborový systém

Katedra softwarového inženýrství

Vedúcí bakalárskej práce: Mgr. Viliam Holub
Študijný program: Správa počítačových systémů

2006

Ďakujem Viliamovi Holubovi a Tomášovi Rosovi za pripomienky k algoritmu.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičaním práce a jej zverejňovaním.

V Prahe dňa 19.5.2006 Ivor Kollár

Obsah

Názov práce: Steganografický souborový systém
Autor: Ivor Kollár
Katedra (ústav): Katedra softwarového inženýrství
Vedúcí bakalárskej práce: Mgr. Viliam Holub
e-mail vedúceho: holub@nenya.ms.mff.cuni.cz

Abstrakt: V predloženej práci študujeme možnosti ochrany informácie v situácii, keď samotná kryptografia nestačí. Skúsime rozobrať možnosť utajenia informácie, nie však v nevinne sa tváriacom nosiči(zvuk, obraz, video), ale v pseudosteganografickom archíve. Pokúsime sa popísať, a neskôr aj implementovať algoritmus, ktorý bude schopný vytvoriť archív, spĺňajúci nasledujúce vlastnosti: Bude obsahovať N súborov, každý chránený jedným z N kľúčov, pričom číslo N sa pokúsime útočníkovi zatajiť. Pri použití jedného z kľúčov získame práve jeden súbor, pričom o existencii ostatných súborov by sme nemali získať žiadnu informáciu. Implementácia tohto algoritmu by mala byť čo najjednoduchšia, platformovo čo najmenej závislá, a jednoducho prakticky použiteľná.
Kľúčové slová: steganografia, kryptografia, archív, súborový systém

Title: Steganographics filesystem
Author: Ivor Kollár
Department: Department of Software Engineering
Supervisor: Mgr. Viliam Holub
Supervisor's e-mail address: holub@nenya.ms.mff.cuni.cz

Abstract: In the present work we study protecting of information in situations, when cryptography alone is not enough. We will try to analyze possibilities of information hiding, not in innocent looking carrier of information(sound, picture, video), but in pseudosteganographic archive. We will try to describe (and later implement) algorithm, that will be able to create archive, meeting following conditions: Archive will contain N files, each protected by one key. Number N must remain hidden to attacker. By using one of keys, we will get only one file, getting no information about rest of files in archive. Implementation of this algorithm should be as simple as possible, maximally platform independent, and easily practically usable.
Keywords: steganography, cryptography, archive, filesystem

Predslov

V tejto práci sa budeme zaoberať programom "stegeek". Stegeek je veľmi jednoduchý jednoúčelový program, ktorý sa pokúša umožniť komunikáciu aj v prostredí, kde útočník môže ku každej poslanej správe získať kľúč(vynútiť si ho). Stegeek využíva pseudosteganografické metódy, nie je to však steganografia ako ju poznáme, teda schovávanie informácie do štandardne vyzerajúcej správy. Je to pokus zatajiť, koľko informácie v skutočnosti prenášame.

1. Kapitola tejto práce rozoberá problematiku z hľadiska obecnejšieho a netechnického, vysvetlíme aké sú naše predpoklady, a čo by mal náš systém vlastne robiť.

2. Kapitola objasňuje čo je to vlastne steganografia, steganalýza, aké techniky používajú, aké nosiče informácii..

3. Kapitola analyzuje náš problém z algoritmického hľadiska, navrhneme samotný algoritmus, a to pre vytvorenie archívu, aj pre extrakciu z neho.

4. Kapitola bližšie rozoberá možné útoky na náš systém v algoritmickej rovine.

5. Kapitola ukazuje konkrétnu implementáciu nášho algoritmu, použité šifry, ako sme sa vysporiadali s problémom generovania náhodných dát..

6. Kapitola je pre koncového užívateľa. Ukážeme ako s našou implementáciou narábať, teda ako vytvárať a extrahovať z archívu, čo všetko môžeme programu špecifikovať, a na čo si dať pri vytváraní archívu pozor.

7. Kapitola zhŕňa, nakoľko sa nám podarilo splniť naše pôvodné stanovené ciele.

Súčasťou práce je priložený CD nosič s programom.

Kapitola 1

Úvod

1.1 Úvod

Našim cieľom je znemožniť určenie, akú časť archívu tvoria informácie, a ktoré dáta to sú. Podobnou cestou sa vydal napr. projekt StegFS (viz. [1]), ktorý má však ambície byť náhradou klasického súborového systému, na úrovni OS. Zameriava sa tým pádom viac na výkon a integráciu s OS. My by sme chceli fungovať v userspace a byť maximálne jednoduchý, jednoúčelový a ľahko použiteľný. Náš systém nazveme "stegeek". Náš prístup k problému nie je priamo steganografia. Minimálne nie v zmysle, ako sme si ju zadefinujeme ďalej. Nepoužijeme žiaden nevinne sa snažiaci vyzerať nosič dát.

1.2 Predpoklady

Majme dvoch ľudí, Alicu a Boba. Alica bude chcieť poslať Bobovi správu, ktorej obsah sa má dozvedieť iba Bob. Avšak, máme tu aj Oskara, ktorý ma záujem správu získať, a ma buď legislatívne (nevydanie kľúča môže slúžiť ako "dôkaz", napríklad aj fingovaného obvinenia), prípadne viac fyzické ("kryptoanalýza gumenou hadicou") páky, ktorými sa môže úspešne pokúsiť kľúč k správe získať.

1.3 Ciele

Chceme popísať (a neskôr aj implementovať) systém, ktorý umožní Alici a Bobovi bezpečne komunikovať a vymieňať si utajené informácie napriek

tomu, že ku každej správe ktorú odošlú budú musieť vydať kľúč. Chceme toho dosiahnuť vytvorením jednoduchého steganografického mechanizmu. Ten dostane na vstup dvojice (súbor, kľúč k priečinku), požadovanú mieru redundancie, a na ich základe vytvorí steganografický archív. Po aplikovaní jedného kľúča na archív sa dostaneme iba k jednému priečinku, pričom by sme (v ideálnom prípade) nemali mať žiadnu možnosť zistiť existenciu ostatných priečinkov. Systém by mal byť prakticky použiteľný, napr. pre e-mailovú komunikáciu. Skladanie viacerých súborov do jedného a kompresiu, (tar, gzip,...) prenecháme na užívateľa, keďže k tomuto účelu sú už vyvinuté ďaleko dokonalejšie nástroje, než by som bol schopný vyvinúť. Samozrejmosťou je, že všetky algoritmy by mali byť verejné, a možnosť dostať sa k priečinku by mala záležať iba na znalosti kľúča.

1.4 Opozícia

Našim protivníkom sa myšlienka, že nebudú vedieť či majú k dispozícii všetky informácie ktoré hľadajú pravdepodobne páčiť nebude. V prípade že majú k dispozícii legálne páky k vydaniu kľúča, pravdepodobne na tomto stave nič nezmenia pri už zachytenom archíve, pretože zákony nie je možné meniť retroaktívne. Do budúcnosti by mohli teoreticky obmedziť dĺžku kľúča, ako sa o to pokúsila istá vláda, ale kontrolovanie dodržiavania takéhoto obmedzenia je minimálne problematické, navyše u kriticky rozmýšľajúcej verejnosti vyvoláva značnú nevôľu. Teoreticky by sa mohli pokúsiť zakázať steganografické techniky ako také, ale to by muselo nutne zasiahnuť aj watermarking, a teda táto možnosť je krajne nepravdepodobná, keďže by išla aj proti záujmu verejnosti, aj proti záujmu korporácii.

V prípade že by protivník použil kryptoanalýzu gumenou hadicou (orig. "Rubber-hose cryptoanalysis"), je situácia trochu menej optimistická. Na jednej strane môže užívateľovi pomôcť vydržať "výsluch" vedomím, že mučiteľ nemôže vedieť či mu užívateľ hovorí pravdu alebo nie. (Teda nemôže to vedieť metódou založenou na analýze nášho archívu) Zároveň, i v prípade že užívateľ nevydrží, a heslo vyzradí, útočník pravdepodobne dlhšiu dobu nebude môcť vyvrátiť podozrenie, že získaná informácia nie je tá, ktorú skutočne hľadá.

Druhou stranou mince je fakt, že keď náhodou užívateľ náhodou zmení názor, a začne chcieť spolupracovať, zo strany útočníkov to môže byť vcelku logicky považované iba za úskok s cieľom vyhnúť sa ďalšiemu mučeniu, a teda budú vo viere v neexistujúce tajomstvo úbohého užívateľa trápiť ďalej. Táto

dvojsečnosť však bohužiaľ nemôže byť eliminovaná, pretože stegeek chráni užívateľa v prvom rade pred slabosťou seba samého. Treba ho však používať s rozvahou, pretože môže priniesť užívateľovi značne väčšiu škodu zo strany represívnych orgánov, než keby sa rovno priznal k svojej, z ich pohľadu ilegálnej činnosti.

Kapitola 2

O steganografii

2.1 Čo je to steganografia

Cieľom steganografie je utajiť existenciu správy. Je príbuznou kryptografie a zdieľajú množstvo spoločných prístupov, ale zatiaľ čo pre kryptografiu je neprijateľné vyzradenie obsahu správy, pre steganografiu je neprijateľné aj samotné zistenie že správa existuje. S steganografiou je veľmi blízko spojený aj watermarking, ktorý sa však skôr zameriava na odolnosť označených dát,(nemožnosť watermark odstrániť), pretože u problematických dát sa už jeho prítomnosť predpokladá. Odhaľovaním steganografie sa zaoberá steganalýza.

2.2 História steganografie

Prvý doložený prípad použitia steganografie sa datuje do k Grécko-Perzským vojnám, keď mal otrok ktorému oholili hlavu, vytetovali na ňu správu, a potom hlavu znova nechali zarásť vlasmi, varovať Grékov pred plánmi perzskej invázie. Zvykli sa používať aj drevené doštičky, ktoré sa po vyrytí správy zaliali voskom, takže vyzerali ako nepoužité. Veľmi starý je aj trik s používaním iba prvých (druhých, tretích...) písmen slov v texte. Neskôr sa používal neviditeľný atrament, mikrobodky... Vyspelejšie steganografie začínajú smerovať k systému, kde nestačí poznať postup, ale je potrebné aj tajné heslo na zistenie prítomnosti skrytej správy.

2.3 Základné techniky steganografie

Nasledujúce techniky nemusia byť nutne disjunktné.

Least Significant Bit (LSB) (najmenej významný bit) je stratégia maskovania, ktorá využíva malé, pre ľudské senzory zanedbateľné zmeny pre ukladanie svojich informácii. Najmenej významný bit preto, pretože keď si zoberieme napr. BMP obrázok (bitmapu) , tak každý pixel je reprezentovaný istou hodnotou farby. Ak to bude napríklad jeden byte, (tj. 256 možných farieb), to máme 8 bitov, a my budeme meniť iba ten najmenej významný, čiže posunieme farbu maximálne o 1. V tomto prípade by to pravdepodobne ešte bolo ľudským okom postrehnuteľné, ale pri napr. 24 bitových farbách už nie. Je to pravdepodobne najzákladnejšia technika digitálnej steganografie.

Kvantizácia DCT koeficientov Koeficienty diskrétnej kosínusovej transformácie sú počas komprimovania súboru mierne upravené (napríklad LSB metódou). Často používaná pri schovávaní informácii do JPEG súborov.

Pertubed quantization ("Zmätočná kvantizácia") Používa sa pri schovávaní dát do komprimovaných formátov, dáta však nechovávame priamo do výsledného komprimovaného súboru, ale upravíme si kompresný algoritmus. Výsledkom sú lepšie štatistické vlastnosti, teda aj lepšia odolnosť voči steganalýze.

One-time pad Steganografická obdoba Vernanovej šifry v kryptografii, teoreticky neprelomiteľná šifra, ktorá vytvára zašifrovaný text, ktorý je nerozlíšiteľný od náhodného textu. V praxi ňou vieme zatajiť, či sme poslali dáta alebo náhodný šum, avšak už samotné poslanie náhodného šumu nás uvrhne do podozrenia, ktorému sa chceme vyhnúť.

2.4 Steganalýza - prirodzený nepriateľ

Steganalýza si kladie za úlohu odhaľovať použitie steganografie na zachytených dátach. Väčšinou sa zakladá na pozorovaní, že pridaním steganografickej informácie k nosiču sa porušia isté štatistické vlastnosti nosiča. Tu sú niektoré základné techniky.

Štatistická analýza (napr. analýza histogramu) Pri vymieňaní poradia bitov(ktoré používajú niektoré techniky, aby čo najmenej zmenili štatistické vlastnosti nosiča), sa niektoré štatistické vlastnosti dvojíc bitov (pravdepodobnosť výskytu hodnoty v aspoň jednom z koeficientov) menia. Celkove existuje veľmi veľa štatistických metód útokov, viacmenej až na útok silou a kontroly kompatibility sa všetky metódu steganalýzy opierajú o pravdepodobnosť. Viac na [5].

RS analýza Pokúša sa odhadnúť počet pixlov v obrázku, ktoré majú zmenený LSB, zvyčajne kvôli primitívnej LSB steganografii. Viac informácii na [7].

Kontrola kompatibility Pri modifikácii komprimovaných formátov ako JPEG, MP3 sa môže stať, že vznikne výsledok, ktorý síce spĺňa daný formát, avšak nemohol vzniknúť ako produkt korektného algoritmu, alebo aspoň nie verejne známeho, čo je už dôvod na podozrenie a podrobnejšie skúmanie daného súboru.

Útok silou Pokiaľ máme podozrenie, že daný súbor v sebe nesie utajenú informáciu, môžeme skúsiť odhadnúť použitý algoritmus (v súčasnosti reálne používaných algoritmov ešte nie je priveľa), a snažiť sa napr. slovníkovými útokmi zistiť kľúč. Tu je situácia rovnaká ako v klasickej kryptografii.

2.5 Typické nosiče informácii

Obrázky Najľahšie, ale v praxi málo používané je schovávanie informácie do bmp obrázku. Málo používané je preto, že bmp obrázky sa kvôli svojej veľkosti bežne cez sieť neposielajú, ale komprimujú sa. Napr. ako JPEG. JPEG (ostatne ako skoro každý komprimovaný formát) je z hľadiska steganografie pomerne ťažkým problémom. Kompresia sa totiž snaží všetky redundantné, príp. nie príliš podstatné dáta eliminovať, steganografia ich však potrebuje pre svoje ciele. Keďže však kompresia nefunguje na 100

Hudba Situácia ohľadom hudby je podobná ako pri obrázkoch, s wav súborom sa ľahko manipuluje, má dostatočnú redundanciu, ale v praxi sa nedá príliš použiť. Pri MP3/Ogg formáte je situácia podobná ako pri JPEGu, avšak celá scéna je oveľa menšia ako pri obrázkoch.

Video Scéna video stenografie je skôr v experimentálnom štádiu, teoretické využitie by bolo pri potrebe schovať obrovské množstvo dát. Zatiaľ sa skôr preberajú trendy z ostatných oblastí. Istý pokrok je sa dosahuje v oblasti watermarkingu (hudby aj videa), keďže táto oblasť je komerčne pomerne zaujímavá.

Text (plain text) Pomerne častý prakticky používaný spôsob steganografie. Využívajú sa hlavne počty medzier, interpunkcia... Problémom zvykne byť nie príliš veľká kapacita nosného textu. Situácia je oveľa lepšia pri html dokumentoch, pretože tam existuje oveľa viac možností ako syntakticky rozdielne zapísať rovnako vyzerajúce dokumenty. Zaujímavý nápad implementuje [8], ktorý ako nosič informácie používa spam. Vzhľadom na množstvo spamu je takto možné prakticky bez podozrenia poslať mail priamo adresátovi.

Iné techniky maskovania Veľmi časté sú aj pseudo steganografické metódy, ako schovávanie informácii do hlavičiek, za oficiálny koniec súboru, atď... Keďže sú ale odhaliteľné každému, iba s znalosťou algoritmu, nebudeme sa nimi podrobnejšie zaoberať, aj keď ich existuje obrovské množstvo.

Kapitola 3

Algoritmus

3.1 Analýza problému

Stojíme pred problémom, ako vydať kľúč k správe, a zároveň nevyzradiť to, čo nechceme. Klasická steganografia rieši tento problém tak, že schová naše tajné dáta do nosiča informácie (text, obrázok..), a spolieha sa na to, že ich útočník nevypátra. Tento systém sa vynikajúco osvedčil, pokiaľ Alica a Bob nie sú priamo monitorovaný, teda Oskar monitoruje sieť ako takú(napr. internet), a snaží sa nájsť znaky použitia steganografie.(a teda pravdepodobne citlivé informácie) Steganalytické metódy v súčasnosti používané (resp. mne v súčasnosti známe) majú príliš veľký počet falošných poplachov. Teda množstvo správ označených ako obsahujúce steganografiu je voči správam skutočne obsahujúcim steganografickú správu neúmerne veľké. V kombinácii s obrovským množstvom dát pohybujúcich sa po sieti a veľmi malým počtom steganografickú informáciu nesúcich dát, tieto metódy nevedú k praktickým výsledkom. Tento systém má však slabšiu odolnosť, ak Oskar sleduje priamo Alicu a Boba. Tí negenerujú príliš veľké množstvá dát, a teda odhaľovanie steganografie sa stáva prakticky použiteľným. Samozrejme za predpokladu že Oskar pozná steganografický algoritmus použitý Alicou a Bobom. Odhalenie použitia steganografie síce ešte neimplikuje znalosť samotnej správy, keďže tá je ešte pravdepodobne zašifrovaná konvenčnou kryptografiou, ale Oskar už vie k čomu kľúč vymáha, a čo by asi(podľa veľkosti) mohlo byť vo vnútri.

My na to skúsime ísť inak. De facto sa budeme odvolávať na neodlíšiteľnosť náhodných dát od zašifrovaných. To otvára otázku, že čo má náš systém spoločne s steganografiou. Prečo by sme nemohli iba zobrať 10 súbo-

rov, všetky zašifrovať, a iba niektoré by obsahovali dáta? V princípe mohli, avšak veľkosti týchto súborov by ostali útočníkovi známe. Z nich by mohol mnohé odvodzovať. Navyše, takto by mal pred sebou útočník konečný počet (10) kľúčov, ktoré by sa snažil získať, aj keď by to napríklad nebolo možné. To sa nám nepáči. Chceli by sme, aby pridané súbory vytvorili čo najhomogénnejšiu masu, pri ktorej by sa bez znalosti konkrétneho kľúča nedalo určiť či daný bajt patrí danému súboru. V ideálnom prípade by sa bez znalosti kľúča nemalo dať určiť vôbec nič. To nás automaticky zvádza použiť hashovaciu funkciu. Tá by dokázala rozdistribuovať vstupné súbory relatívne rovnomerne po celom archíve. Prázdne miesta by sme si mohli vyplniť na koniec. To od nás však vyžaduje dopredu poznať veľkosť výstupného archívu. Ta sa bude určite skladať zo súčtu všetkých vstupných súborov krát tajomná konštanta hashovacej funkcie, aby sme nemali priveľa kolízii. To však stačiť nebude. Útočník by si jednoducho odčítal veľkosť už známych súborov od veľkosti archívu, a zistil by koľko dát mu ešte chýba. Preto musíme pridať istú redundanciu. Veľkosť tejto redundancie musí ostať útočníkovi neznáma. Táto redundancia bude kľúčovým bodom nášho algoritmu, a práve neznalosť jej veľkosti bude maskovať citlivé dáta.

Táto myšlienka vyzerá na prvý pohľad primitívne, veď si iba povieme že napr. 30% dát je redundancia, a útočník nám to má uveriť? Útočník však nebude mať inú možnosť, pokiaľ nám nebude schopný dokázať opak. A to by bez znalosti kľúča skrývajúceho ďalších napr. 5% dát nemal byť schopný. Faktom ostáva, že útočník bude musieť "Pristúpiť na našu hru s steganografiou", ale ako sme rozoberali vyššie, ťažko nám to môže zakázať.

Pred nami stojí problém, ako dáta pridať do archívu, aby sme Oskarovi nenechali žiadnu stopu o ich pôvode, príslušnosti k súboru či nebodaj náhodnom pôvode. Tieto chceme utajiť aj v momente keď už bude Oskar poznať N-1 kľúčov. Nesmie byť teda schopný rozlíšiť stav keď bude poznať N a stav keď bude poznať N-1 kľúčov. Nech už pozná koľko chce kľúčov, hoc aj N, hodnota N, aká bola špecifikovaná pri vytváraní archívu mu musí ostať utajená. Nie je to až taký triviálny problém, ako by sa mohlo zdať.

Dáta si rozdelíme na bloky. Bude sa nám s nimi ľahšie pracovať, a budeme ich môcť rozhadzovať pomocou hashovacej funkcie. Kľúč podľa ktorého budeme blokom určovať indexy v archíve bude tajný kľúč, ktorý nám chráni prístup k konkrétnemu súboru v rámci archívu. Samozrejme, bloky by sme mali aj šifrovať, ale to teraz neberieme v úvahu. V prípade že dôjde ku kolízii, teda vygenerujeme index, ktorý je už obsadený, jednoducho ho preskočíme, a vygenerujeme index ďalší.

Keby sme napríklad súbory pridávali vyššie uvedeným spôsobom, teda rozhádzaním súborov do archívu a nakoniec zaplnením voľných miest, počet kolízii pri extrakcii všetkých súborov, keď nerátame kolízie medzi samotnými súbormi by bol nula, a teda by sme úplné triviálne zistili či máme všetky súbory z archívu. Redundanciu budeme teda musieť pridávať naraz s ostanými dátami. Nemôžeme však pridávať súbory postupne po sebe. V tom prípade by počet kolízii pri extrakcii prvého pridaného súboru (aj keby sme redundanciu dali úplné prvú) bol štatisticky nižší ako pri poslednom pridávanom súbore. To by nás mohlo pomerne spoľahlivo odhaliť. Preto musíme súbory pridávať po jednom bloku, tj. Jeden blok z jedného súboru, ďalší blok z ďalšieho, atď...

Nemôžeme však súbory a redundanciu pridávať po sebe v akomkoľvek poradí. Ak by sme to robili, dali by sme Oskarovi možnosť odlíšiť stav keď pozná N a N-1 kľúčov. Bloky budeme musieť pridávať v náhodnom poradí. Našu schopnosť ich extrahovať z archívu to neovplyvní, pretože pri extrakcii ich spoznávame podľa indexov generovaných z nášho kľúča hash funkciou. Na mechanizmus rozpoznania či daný blok patrí nám využijeme to, že sme blok predtým zašifrovali, a na začiatok bloku sme dali naše id. (pred zašifrovaním samozrejme, v rámci jedného archívu musia byť všetky id unikátne). Tento systém už začína byť pomaly použiteľný.

Musíme dať ale ešte pozor na to, s akou pravdepodobnosťou budeme pridávať redundanciu, ak budeme vyberať medzi blokmi náhodne. V momentne keď by sme jej totiž priradili rovnakú pravdepodobnosť ako ostatným súborom, mohol by útočník rozlíšiť nasledujúce 2 situácie (v špeciálnom prípade, že všetky súbory majú rovnakú veľkosť):

Pozná N(počet súborov v archíve) kľúčov. V tom prípade, napriek náhodnému pridávaniu blokov, pravdepodobnosť nálezu kolízie, ktorá neprináleží žiadnemu zo známych súborov bude 1/N+1.

Pozná N-1 kľúčov. V tomto prípade bude tá istá pravdepodobnosť rovná 2/N+1. (Pretože boli pridávaný redundancia a aj jeden regulárny súbor)

Samozrejme predpoklad že všetky súbory majú rovnakú veľkosť je silne obmedzujúci, a rozhodne by to nebola obecná schéma útoku. Pracujeme však s blokmi pevnej veľkosti, a veľkosti súborov sa tým pádom zaokrúhľujú. Preto by tento, ale jemu dostatočne podobný prípad mohol nastávať nepríjemne často.

3.2 Návrh

Výsledný produkt nášho myšlienkového postupu teda bude vyzerať takto: Zoberieme N súborov, pridáme N+1vý (pseudonáhodne generované dáta). Súbory si rozdelíme na bloky. Pomocou ďalšej pseudonáhodnej postupnosti a hashovacej funkcie, budeme vyberať náhodne vždy jeden zo súborov, a jeho prvý blok, ktorý budeme mať k dispozícii použijeme. Všetky súbory majú rovnakú pravdepodobnosť vybratia, redundanciu vyberáme s pravdepodobnosťou inou. V každom kroku vyberieme jeden zo súborov, zoberieme jeho blok, ten zašifrujeme prináležiacim kľúčom. Kľúč následne zmutujeme, a použijeme na určenie indexu v primárnom súbore. Ak vybrané miesto nie je voľné, tento krok opakujeme, pokiaľ to voľné miesto nenájdeme. Zašifrovaný blok do primárneho súboru zapíšeme. Takto pokračujeme až pokiaľ sa nám jeden zo súborov neminie, teda nebude už mať žiadne bloky na zápis. Tento súbor potom vylúčime spomedzi kandidátov na zápis, a teda sa zmenia pravdepodobnosti vybratia ostatných súborov. Takto pokračujeme, až pokiaľ nezapíšeme všetky regulárne súbory. Na koniec prejdeme primárny súbor, a vyplníme náhodnými dátami všetky nezaplnené bloky.

Pri extrakcii budeme z kľúča generovať dešifrovacie podkľúče a podkľúče určujúce indexy v primárnom súbore. V prípade, že narazíme na blok, ktorý sa nedá dešifrovať aktuálnym dešifrovacím podkľúčom, vygenerujeme ďalší indexový podkľúč, a tento blok ignorujeme (teda nastala kolízia pri extrakcii). V prvom bloku, ktorý úspešne dešifrujeme máme zapísanú veľkosť súboru, ktorý extrahujeme. Tieto kroky opakujeme dovtedy, kým nenazbierame dostatočný počet bytov.

3.3 Definície

Majme súbory otvoreného textu $P_1...P_N$, kde N je číslo známe iba Alici.

Ku každému z $P_1...P_N$ Alica dodá/vytvorí jeden kľúč, teda máme $K_1...K_N$. Ku každému kľúču si navyše budeme pamätať 4 bytové id, všetky id v rámci jedného archívu musia byť unikátne. Samotné id nepovažujeme za súčasť kľúča pri algoritme šifrovania, budeme ho však považovať za súčasť kľúča z pohľadu používateľa (ďalšie 4 byty k dĺžke kľúča).

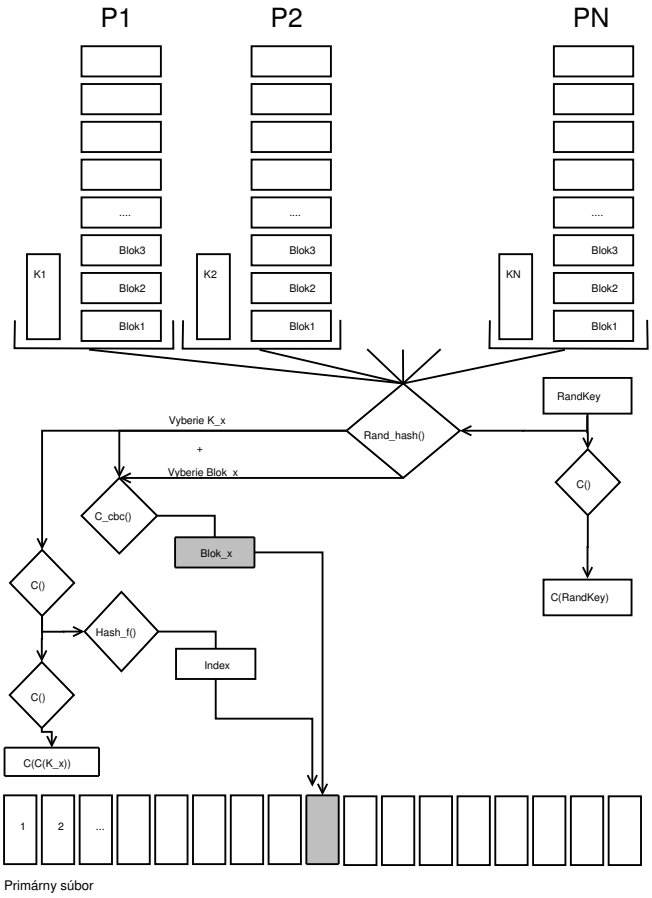
Alica si ešte zvolí redundanciu R, teda koľko percent výsledného archívu bude tvoriť pseudonáhodná výplň. Toto číslo ostane známe len jej, a po vytvorení archívu ho zničí.

C_cbc_Mode(A,B,C,D) je cbc mód šifrovacieho algoritmu, ktorý zašifruje

blok dát A do bloku dát B kľúčom C, s veľkosťou bloku A = D. C(A,B,C) zašifruje blok natívnej dĺžky algoritmu A (napr. 256 bitov) bo bloku B kľúčom C.

3.4 Vytvorenie archívu

Jeden krok algoritmu vyváraniа archívu, v prípade že nenastane kolízia.



Obrázek 3.1: Jeden krok algoritmu

Vypočítame si výslednú veľkosť archívu
$OUTPUT\_SIZE = (\sum_{i=1}^{N} P_i) * kontanta\_hash\_funkcie * redundancia$

Konštanta hash funkcie zabezpečí, aby sme nemali priveľa kolízii pri tvorbe archívu.

Veľkosť archívu počítame z veľkosti P_i zaokrúhlených na celočíselný násobok BLOCK_SIZE. Ešte si N zvýšime o 1. Nové P_N nám bude označovať redundantné dáta pridávané do archívu.

Samotný archív:

Rozdelíme si P_i na bloky veľkosti BLOCK_SIZE-4. Na začiatok prvého bloku každého P_i súboru dáme id P_i(4 byte) + veľkosť bloku (8 byte) (teda bude obsahovať iba BLOCK_SIZE-12 byte dát).
Na začiatok ostatných blokov dáme iba id P_i.

Vygenerujeme si náhodný kľúč RANDGEN. S týmto kľúčom budem následne vykonávať operácie:

- C(RANDGEN,RANDGEN,RANDGEN) (teda zašifrujeme kľúč sám sebou znova do seba)
- Skopírujeme 4 byty z RANDGEN
- spravíme modulo (N*R) na skopírované 4 byty

Robíme modulo N*R preto, pretože redundantné dáta majú vyššiu pravdepodobnosť výskytu, než len jeden z otvorených textov. Ak nám vyjde číslo väčšie než N-1, znamená to, že sme náhodne vybrali redundantné dáta. Tým budeme dostávať pseudonáhodnú postupnosť, nepredpokladateľnú bez znalosti RANDGEN kľúča. Táto postupnosť nám určí vždy jeden z $P_1...P_N$.

Takže v každom cykle:

1. Vyberieme náhodne jeden z $P_1...P_N$
   dostaneme teda jeden BLOK dát, a K_i, k nemu prináležiaci kľúč.
2. C_cbc_mode(BLOK,BLOK,K_i,BLOCK_SIZE)
   zašifrujeme BLOK pomocou k nemu prináležiaceho kľúča v cbc móde šifry
3. C(K_i, K_i, K_i)
   Vygenerujeme si nový čiastkový kľúč

4. Index = (4 byte z K_i mod (OUTPUT_SIZE/BLOCK_SIZE))
   spočítame si index, ktorý nám bude ukazovať na jeden z blokov vo výstupnom súbore

5. Prečítame si, čo sa nachádza v danom bloku vo výstupnom súbore(na prvých 4 byte-och). Ak je tam 0, ešte sme tam nič nezapisovali, a teda blok je voľný. Ak voľný nie je, opakujeme predchádzajúce 2 kroky dovtedy, kým voľný blok nenájdeme (teda 3. a 4.)

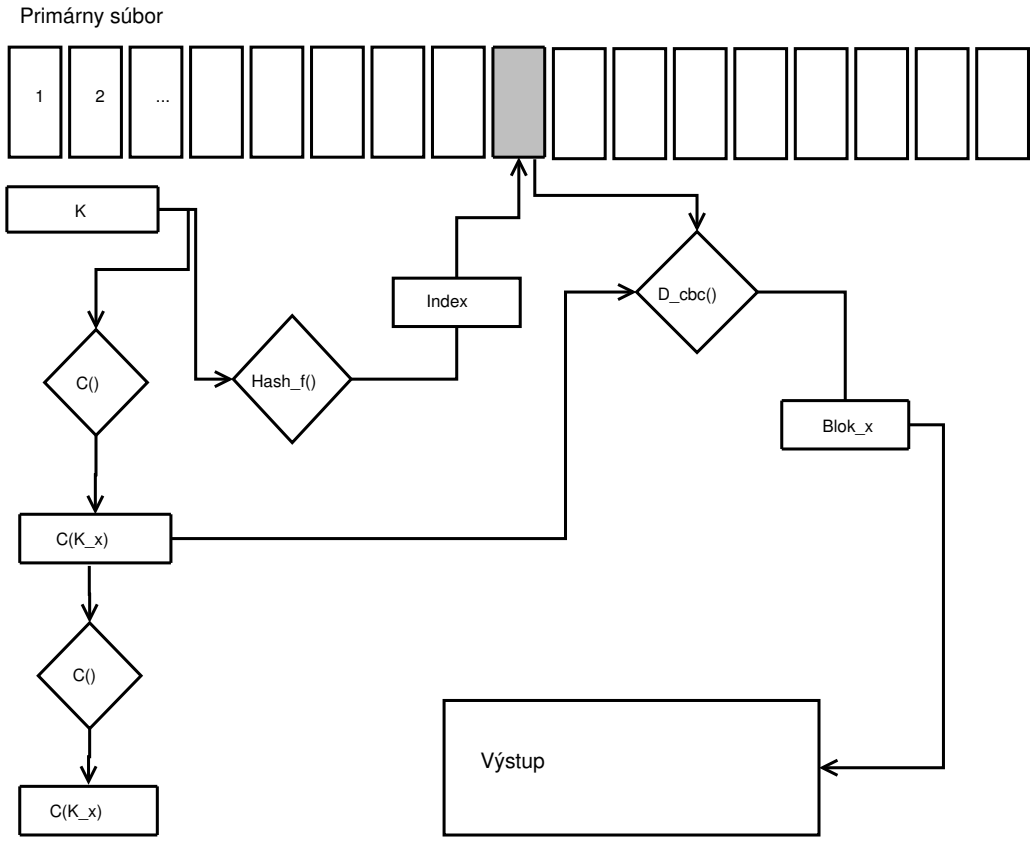
6. Zapíšeme BLOK na vybranú pozíciu do výstupného súboru. Pripravíme si ďalší blok z P_i (pretože tento sme už spracovali)

Kroky 1 až 6 opakujeme, pokiaľ máme aspoň v jednom zo súborov $P_1 ... P_N - 1$ dáta na zapísanie.

Dostaneme sa do stavu, keď sme zapísali všetky dáta do archívu. Ostanú nám však ešte voľné miesta (samé nuly) vo výstupnom súbore. Tie teda doplníme pseudonáhodnými dátami.

3.5 Extrakcia z archívu

Jeden krok algoritmu extrakcie, za predpokladu že nenastane kolízia.



Obrázek 3.2: Jeden krok algoritmu

Mám archív C a kľúč K. Posledné 4 byty z K tvoria id, oddelíme ich a za K považujeme iba zvyšok.

- Zapamätáme si kľúč K ako dešifrovací kľúč pre blok. (DECKEY)
- C(K,K,K) // Vygenerujeme si nový čiastkový kľúč
- Index = (4 byte z K_i mod (INPUT_SIZE/BLOCK_SIZE)) // spočítame si index, ktorý nám bude ukazovať na jeden z blokov vo výstupnom súbore.

Opakujeme tieto 2 kroky podobne ako pri vytváraní archívu, ale iba dovtedy, kým získaný blok po dešifrovaní kľúčom DECKEY nebude mať na prvých 4 bytoch hodnotu id.
Ak sme našli náš prvý blok, zistíme si z neho celkovú veľkosť nášho schovaného P_i. Ak sme predchádzajúce kroky vykonalo INPUT_SIZE/BLOCK_SIZE počet krát(pripadne iný, zatiaľ presne neodhadnutý počet krát, pri ktorom pravdepodobnosť že ide o správny kľúč, je dostatočne blízko nule), znamená to že kľúč nie je správny.

-1. C(K,DECKEY,K) // Vygenerujeme si nový čiastkový kľúč, a zapamätáme ako dešifrovací kľúč

-2. C(K,K,K) // Vygenerujeme si nový čiastkový kľúč

-3. Index = (4 byte z K_i mod (INPUT_SIZE/BLOCK_SIZE)) // spočítame si index, ktorý nám bude ukazovať na jeden z blokov vo výstupnom súbore

-4. D(BLOK,BLOK,DECKEY) // dešifrujeme blok

Kroky 2, 3, 4 opakujeme, až kým nenájdeme BLOK patriaci nám. (teda po dešifrovaní pomocou DECKEY bude na prvých 4 bytoch naše id), potom sa vrátime zase na 1.

Celý cyklus opakujeme, pokiaľ nenazbierame počet bytov, ktoré sme si zistili z nášho prvého bloku.

Kapitola 4

Možné útoky

4.1 Bruteforce

V prípade, že by bol útočník schopný vyskúšať všetky možné kľúče, ochranu archívu samozrejme prelomí. Tomu sa pravdepodobne nedá zabrániť pri žiadnom systéme. Je už na implementácii algoritmu, aby sme útočníkovi neponúkli žiadnu duplicitnú informáciu, ktorá by mohla útok zjednodušiť.

4.2 Pseudonáhodná postupnosť

Nebezpečnou cestou útoku sa javí byť útok na pseudonáhodnú postupnosť. Nie preto, že by to bol problém teoreticky neriešiteľný, ale preto, že je veľmi náchylný na implementačné chyby. Nechcené vyzradenie i malej časti kľúča tejto postupnosti môže mať za následok prelomenie ochrany celého archívu.

4.3 Štatistická analýza

V našom prípade by to bola štatistická analýza založená na počte kolízii ku ktorým dôjde počas extrahovania súboru z archívu. Napríklad, keby sme súbory nepridávali po blokoch náhodne, ale v istom poradí, pravdepodobnosť kolízie pri extrahovaní jedného súboru by bola odlišná od pravdepodobnosti kolízie pri extrahovaní druhého súboru, a dalo by sa určiť poradie, k ktorom boli súbory do archívu pridávané, a teda aj predpokladať existencia iných súborov. Tento príklad možno vyzerá vykonštruovane, ale v skutočnosti ešte

môžu byť v našej schéme takéto rôzne pravdepodobnosti zakomponované, aj keď nie takto viditeľne.

Útokom vedeným cez štatistické vlastnosti dát archívu by sme sa mali vyhnúť, pretože i náhodné dáta sú zašifrované tou istou implementáciou algoritmu, ako dáta ostatné.

4.4 Možný problém s redundanciou

Podozrenie na možnosť predchádzajúceho útoku je pri implementácii redundancie. Keď máme vyberať náhodne z N+1 blokov jeden ktorý pridáme, akú má mať pravdepodobnosť vybrania N+1 tého bloku(redundancie)?

Ak by sme jej nechali rovnakú pravdepodobnosť ako ostatným blokom (tj. 1/N), útočník by v prípade že pozná všetky kľúče mal pravdepodobnosť kolízie 1/N, (keď neráta kolízie s súbormi, ku ktorým má kľúče). Keby však poznal iba N-1 kľúčov, predchádzajúca pravdepodobnosť mu vyjde v závislosti od veľkosti ukrytého súboru a dĺžky archívu o niečo vyššia.(podrobnejšie sme tento problém rozobrali v "Analýze problému") Či by išiel tento útok reálne použiť je otázne. Pri malých objemoch dát takmer určite nie, pretože rozdiel v týchto pravdepodobnostiach nie je dostatočne veľký. Pri väčších objemoch dát by však mohol byť reálne použiteľný.

Ostáva však otázka, akú veľkú pravdepodobnosť vybrania by táto redundancia mala mať. Odpoveď je podľa mojich doterajších poznatkov netriviálna. Minimálne to nie je výpočet na jeden riadok programu. Presnosť mojej jednoriadkovej aproximácie je tiež na dlhšiu diskusiu, pravdepodobne presahujúc rámec tejto práce.
Otázka koľko dát je potrebných na odlíšenie aproximácie od skutočnej hodnoty, a tým aj prípadné otvorenie archívu útoku ako v predchádzajúcom prípade ostáva tiež otvorená. Dovolím si tvrdiť že bude potrebné značné úsilie na jej vyriešenie, nakoľko ide o netriviálny problém.

Kapitola 5

Implementácia

5.1 Implementácia

Projekt som napísal v jazyku C, natívne som ho vyvíjal pod LINUXom s prekladačom gcc, kde je jeho fungovanie bezproblémové. Vďaka použitiu "GNU automake" utility, ktorá umožňuje automatické generovanie konfiguračných skriptov, by s jeho kompiláciou a používaním nemali byť problémy na žiadnej UNIX odvodenej platforme. Program bol testovaný na LINUXe, FreeBSD, Solarise. Pre kompiláciu na Windows* platforme je potrebné prostredie CGWYN /citecgwyn, prípadne funkčne podobná alternatíva. Mierne problémy spôsobuje rozdielna implementácia generovania náhody na rôznych operačných systémoch. Preto má užívateľ možnosť špecifikovať zdroj náhody ručne pomocou prepínača --randomsource.

Adresárová štruktúra projektu je nasledovná:

stegeek_x.y.z koreňový adresár projektu obsahuje informácie o projekte(AUTHORS, Changelog, COPYING, NEWS, ...), rýchle informácie technického charakteru (README, INSTALL, ...) a samokonfigurujúce inštalačné skripty. Obsahuje aj nasledujúce podadresáre:

config obsahuje konfiguračné súbory utility "GNU automake".

doc obsahuje dokumentáciu k algoritmu programu. Zatiaľ iba v slovenčine.

example obsahuje sadu skúšobných príkladov, pre demonštráciu funkčnosti programu.

src obsahuje samotné zdrojové kódy. Podadresár aes obsahuje implementáciu AES algoritmu.

Dátové štruktúry týkajúce sa pridávaných súborov (struct onefile) alokujem dynamicky pri spracovaní parametrov volania. Štruktúra onefile vyzerá nasledovne:

```
struct onefile {        // information about file, we are adding

        int fd;
        unsigned int id;
        unsigned long size;
        char keygen[32];
        AES\_KEY key;
        char block[BLOCK\_SIZE];

};
```

5.2 Použité šifry

v našej implementácii používame AES (Rijandel) algoritmus s pevnou dĺžkou kľúča 256 bitov. Samotný kód AES-u v jazyku C je prebratý z OpenSSL projektu. Pôvodne stegeek používal OpenSSL iba ako externú knižnicu, ale tá nie je prítomná v každom systéme, a je pomerne objemná. Tak som vybral iba kód AES algoritmu, a knižnica OpenSSL nie je ďalej používaná. Z hľadiska opravovania chýb a nových verzii to nie je ideálne, ale umožňuje to udržať stegeeka pomerne malého a samostatne fungujúceho. Konkrétne sú používané funkcie AES_encrypt(), AES_cbc_encrypt(), AES_decrypt(), AES_set_encrypt_key(), AES_set_decrypt_key() .

5.3 Generovanie náhodných dát

Náhodné dáta sú generované vo funkcii generate_junk().
Na začiatku máme dva 256 bitové kľúče. Z jedného sa vygeneruje skutočný AES kľúč, dhuhý sa použije ako základ náhodných dát . Skopírujeme ho na začiatok bloku, a zašifrujem ho AES kľúčom z prvého kľúča vygenerovaným. Výsledok znova zašifrujeme týmto kľúčom, ale uložíme na ďalšiu

pozíciu. Takto pokračujeme (s tým istým AES kľúčom), až kým nezaplníme celý blok. Týmto postupom by sme mali dostať bez znalosti kľúča nepredpokladateľnú pseudonáhodnú postupnosť, s úplne rovnakými štatistickými vlastnosťami ako zašifrovaný text.

Kapitola 6

Použitie

6.1 Inštalácia

Program je možné stiahnuť z [9]. Inštalácia prebieha klasickým UNIXovým spôsobom:

Rozbalíme komprimované zdrojové kódy:

```
$ tar -xvzf stegeek_0.9.8.tgz
```

Vojdeme do koreňového adresára programu.

```
$ cd stegeek_0.9.8
```

Spustíme konfiguračný skript. V prípade že nechceme inštalovať program na celý stroj, ale iba pre konkrétneho užívateľa, určíme cestu kam sa spustiteľný súbor nainštaluje pomocou prepínača --prefix. Napríklad takto:

```
$ ./configure --prefix=/home/user/bin
```

V opačnom prípade postupujeme štandardne.

```
$ ./configure
```

alternatívne:

```
$ . configure
```

ak používame csh.
Teraz program skompilujeme.

```
$ make
```

V tomto momente by sme už mali mať funkčný samotný program. Môžeme ho buď ručne skopírovať na želané miesto, alebo to urobiť automaticky:

```
$ su
# make install
# logout
$
```

V prípade že všetko prebehlo v poriadku, sme teraz príkazom

```
$ stegeek
```

schopný program spustiť. Tým je inštalácia hotová.

6.2 Všetky možnosti volania (options)

Obecná syntax volania nášho programu je

```
$ stegeek {prepínače} [vstupné súbory] {< súbor s kľúčmi}
```

Prípustné parametre sú:

-e, --extract Nemá žiaden parameter. Určí, že archív sa nebude vytvárať, ale bude sa extrahovať z— už existujúceho archívu. Konflikt s –redundancy.

-o, --output Má 1 argument, štandardne nastavený na "archive.sgk" . V— prípade že –output nie je špecifikovaný, použije sa štandardný, inak užívateľom zadaný výstupný súbor. Použiteľný pri vytváraní súboru aj extrakcii.

-r, --redundancy Má 1 argument. Používa sa iba pri vytváraní archívu. Určuje, koľko redundantných dát bude vo výslednom archíve. Použije sa formát desatinného čísla, kde 1.00 predstavuje 100% redundanciu. Užívateľ musí špecifikovať redundanciu pri vytváraní archívu. Požívanie porogramu s štandardne nastavenou redundanciou je nezmyselné. Konflikt s –extract.

-s, --randomsource Má 1 argument. Umožňuje užívateľovi špecifikovať alternatívny zdroj náhody. Stegeek teda otvorí zadaný súbor, a prvých 32 bytov použije ako kľúč, z ktorého bude ďalej generovať pseudonáhodné dáta.

-h, --help Nemá žiaden parameter. Vypíše nápovedu (príklad použitia) a— ukončí program. Rovnaké chovanie ako pri zavolaní programu s— nesprávnymi resp. kolidujúcimi parametrami, alebo úplne bez parametrov.

-v, --version Nemá žiaden parameter. Vypíše verziu programu a— skočí. Z— konvenčných dôvodov. V— súčasnosti synonymum –help.

6.3 Vytvorenie archívu

Pri vytvorení archívu potrebujeme zadať súbory ktoré chceme v archíve schovať, a ku každému súboru musíme zadať jeden kľúč. Ešte musíme špecifikovať redundanciu, pretože použiť stegeek-a s štandardnou redundanciou by nemalo zmysel. Príkaz

```
$ stegeek -o archive.sgk -r 2.50 file1 file2 file3
   tajneheslo1
   tajneheslo2
   tajneheslo3
```

vytvorí archív s 250% redundanciou zo súborov file1, file2, file3 a použije k tomu kľúče interaktívne zadané.
Keď vytvárame archív, je veľmi praktické napísať kľúče do jedného súboru (na každý riadok jeden), a poslať ich programu cez presmerovanie `<`
Napríklad tento:

```
$ stegeek -o archive2.sgk -r 0.3 -s /dev/urandom \
  example/file* < example/keyfile
```

vytvorí archív s 30% redundanciou zo všetkých súborov v adresári, ktoré začínajú na "file", a použije k tomu kľúče uložené v súbore example/keyfile. Pseudonáhodná postupnosť sa inicializovala zo súboru /dev/urandom.

Súčasná implementácia potrebuje na jeden archív 128 bytov náhodných dát v súbore, teda keď určujeme alternatívny zdroj náhody, mali by sme sa uistiť, že zadaný súbor toľko dát obsahuje. Pri používaní LINUX-ového

/dev/urandom generátoru ako zdroja náhody sa môže stať, že program na chvílu prestane reagovať. Je to spôsobené tým, že urandom berie náhodné udalosti z vyvolávania prerušené, a v istom okamihu jednoducho nebude mať dosť prerušení na vygenerovanie dostatočného počtu bytov. V takom prípade je vhodné hýbať myšou, prípadne do okna inej aplikácie písať náhodný text. To by malo problém rýchlo vyriešiť.

6.4 Extrakcia z archívu

Pri extrakcii musíme špecifikovať prepínač --extract, ďalej musíme zadať stegeek-ovi archív z ktorého chceme extrahovať, a jeden kľúč. Stegeek nám buď vráti súbor z archívu, prináležiaci tomuto kľúču, alebo nám povie že kľúč nie je správny.

Tento príkaz

```
$ stegeek -e -o extract1.out archive1.sgk < example/secretkey
```

extrahuje z archívu archive1.sgk súbor, prináležiaci kľúču zo súboru example/secretkey do súboru extract1.out .
Tento

```
$ stegeek -e
 12345678901234567890
```

extrahuje zo štandardného archívu (archive.sgk) do štandardného výstupu (extract.out) s interaktívne zadaným kľúčom.

Kapitola 7

Záver

7.1 Záver

Systém popísaný v úvode sa podarilo navrhnúť aj implementovať. Je ľahko prakticky použiteľný. V súčasnosti mi nie je známy žiadny prakticky použiteľný útok proti stegeekovi. Túto možnosť však nemôžme do budúcnosti vylúčiť, a to buď v prípade, že by zlyhal AES, alebo jeho implementácia, ktorú stegeek využíva, alebo v prípade že by sa našlo kritérium dostatočne citlivé na to, aby odlíšilo približne vypočítanú pravdepodobnosť redundantne pridanej informácie pri vytváraní archívu od pravdepodobnosti skutočnej. To by mohlo teoreticky spôsobiť zraniteľnosť nášho archívu štatistickou analýzou podľa počtu kolízii pri extrakcii.

Dôkaz toho že takéto kritérium neexistuje, resp. nájdenie takéhoto kritéria považujem za problém netriviálny, a vyžadujúci ďalšie skúmanie (v prípadnej neskoršej práci).

Za približne 30 dní od zverejnenia prvej betaverzie na internete [9], som zaznamenal približne 100 stiahnutí programu, a prišli mi 3 oznámenia o nájdených chybách. V súčasnej podobe by program nemal obsahovať žiadnu chybu, ktorá by mohla spôsobiť nekorektné chovanie a ohroziť bezpečnosť dát. (Okrem potencionálnych problémov spomínaných v kapitole "Možné útoky").

Literatura

[1] Andrew D. McDonald, Markus G. Kuhn:
StegFS: A Steganographic File System for Linux
in the 1999 Information Hiding proceedings, Springer-Verlag, LNCS 1768

[2] Ross Anderson: *Stretching the Limits of Steganography*
Information Hiding, First International Workshop, Cambridge, U.K.
May 30 - June 1, 1996, ISBN: 3-540-61996-8

[3] *The Steganographics File System*
`http://www-users.rwth-aachen.de/peter.schneider-kamp/sources/sfs/`

[4] `http://niels.xtdnet.nl/stego/`

[5] Jessica Fridrich, Miroslav Goljan *Practical Steganalysis of Digital Images – State of the Art*
`http://www.ws.binghamton.edu/fridrich/Research/steganalysis01.pdf`
(SUNY Binghamton, Department of Electrical Engineering)

[6] Niels Provos: *Defending Against Statistical Steganalysis*
`http://www.citi.umich.edu/u/provos/papers/defending.ps`
on 10th USENIX Security Symposium, August 2001.
(February 2001, CITI Techreport 01-4)

[7] SUNY Binghamton, Department of Electrical Engineering:
`http://www.ws.binghamton.edu/fridrich/publications.html`

[8] *spamimic* `http://www.spammimic.com/explain.shtml`

[9] *stegeek* `https://www.hysteria.sk/~niekt0/stegeek/`

[10] *Cygwin* `http://www.cygwin.com/`