Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Híradástechnika Tanszék

Szteganográfiai algoritmusok vizsgálata

Készítette: Földes Ádám Máté

E-mail: fa606@hszk.bme.hu

Konzulensek: Gódor Győző (HIT), Gulyás Gábor György (HIT)

2009.

NYILATKOZAT

Alulírott Földes Ádám Máté, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2009. május 15.

.....

Köszönetnyilvánítás

Köszönöm konzulenseimnek, Gódor Győzőnek és Gulyás Gábornak a támogatást, mellyel hozzájárultak a diplomaterv sikeres elkészítéséhez és a nehezebb munkafázisokon történő átlendüléshez.

Kivonat

A szteganográfia az információrejtés tudománya. Olyan módszerek tartoznak ide, melyek nem az információ tartalmát, hanem annak létezésének tényét fedik el a támadók elől. Sokféleképpen lehet hasznosítani egy szteganográfiai algoritmust, de jelen diplomadolgozatban a szteganografikus állományrendszerek koncepciójára fogok koncentrálni.

Javaslatot teszek továbbá a szteganografikus fájlrendszer koncepciójának javítására a „stegodrive"-implementáció formájában, valamint egy saját adatrejtő algoritmust is ismertetek, mely a legjelentéktelenebb bites szteganográfia továbbfejlesztéseként főleg az információrejtés vizuális észlelésével szemben lehet hatékony. Mindemellett részletesen kitérek a különböző szteganográfiai módszerek tulajdonságaira, különös tekintettel azok biztonságára.

Az analízis során feltártam bemutattam az algoritmus biztonságos voltát a vizuális észleléssel szemben, de az is kiderült, hogy a statisztikai szoftverekkel végzett szteganalízissel szemben a módszer nem biztonságos. Javasoltam viszont egy olyan továbbfejlesztést, mellyel a feltárt hiányosságok esetleg kiküszöbölhetőek lehetnek. Mindezek mellett elemeztem a stegodrive mint koncepció szteganografikus biztonságát különböző képességű ellenfelekkel szemben, és bemutattam, hogy bizonyos típusú támadókkal szemben a stegodrive hatékony védekező eszköz, valamint javasoltam vázoltam néhány továbbfejlesztési lehetőséget, melyek segítségével a szoftver még többféle ellenféltől védheti meg az elrejtett információt.

Abstract

Steganography is the science of information hiding. Methods that belong to this discipline concentrate on hiding the very existence of information rather than simply its contents. Application of steganographic algorithms is manyfold, but in this thesis I concentrate on steganographic file systems.

Furthermore, I propose a novel improvement for steganographic file systems in the shape of the 'stegodrive'-implementation, and I discuss an own data hiding algorithm that can be considered an improvement over simple least significant bit steganography in bitmap images, which is most effective at defeating visual detection of hidden information. Besides, I go into details of the properties of various steganographic algorithms, especially their security.

During the analysis I demonstrated the security of the proposed algorithm against visual detection, but I also found out that the method was vulnerable against statistical steganalysis. However, I proposed an improvement with which these deficiencies may be avoided. Furthermore, I analysed the stegodrive as a concept against adversaries of various capabilities, and I demonstrated that the stegodrive is an efficient defence against certain attackers, and I proposed a few possibilities for improvement which could make to software effective against other kinds of adversaries.

Tartalomjegyzék

1. Bevezető **1**
1.1. A szteganográfiával kapcsolatos problémák 2
1.2. A szteganográfia és a privátszféra kapcsolata 3

2. A szteganográfia irodalma **5**
2.1. Csoportosítás tartomány szerint 5
2.2. Csoportosítás adaptivitás szerint 6
2.3. Csoportosítás cél szerint 6
2.4. Szteganalízis 7
2.5. Adatrejtés szövegekbe 8
2.6. Adatrejtés képekbe 9
2.6.1. Adatrejtés raszteres tömörítetlen képekbe 9
2.6.2. Adatrejtés palettás tömörítetlen képekbe 12
2.6.3. Adatrejtés tömörített képekbe 13
2.7. Adatrejtés tömörítetlen hangfájlba 14
2.8. Adatrejtés videofolyamba 15
2.9. Adatrejtés binárisba 16
2.10. Szteganografikus fájlrendszerek 17
2.10.1. StegFS 17
2.10.2. vstegfs 18
2.10.3. TrueCrypt Hidden Volume 18
2.10.4. A szteganografikus fájlrendszerek hatékonysága 18
2.11. Formátum gyengeségek kihasználása szteganográfiai célokra 20

3. A saját szteganográfiai szoftver implementációjának vázlata **22**
3.1. Modell a kapacitás-észrevétlenség kompromisszum egydimenziós leképzésére 22
3.2. Az adatbáziskezelő működése 25
3.3. A grafikus felület 26
3.4. Az adatrejtés megvalósítása 28
3.5. A rétegek implementációja 31

4. Analízis **34**
4.1. A javasolt normaértékes LSB algoritmus elemzése 34
4.1.1. Kapacitás 34
4.1.2. Biztonság 35
4.1.3. Komplexitás 38
4.2. A stegodrive mint koncepció elemzése 39

4.2.1. Teljesítmény 39
4.2.2. Támadó modell 40
4.2.3. Az allokációs stratégia hatása a biztonságra 42
4.3. Az adatrejtés hatása a statisztikai tulajdonságokra 43
4.3.1. A módszertan kiterjesztése 44
4.3.2. A felhasznált minta 45
4.3.3. A statisztikák elkészítésének módja 45
4.3.4. Szteganalízis titkosítatlan rejtett információra 47
4.3.5. A rejtjelezés hatása a szteganalízisre 48
4.3.6. Kiegyensúlyozatlan forrás hatása a szteganalízisre 49

5. Konklúzió **51**
5.1. Továbbfejlesztési lehetőségek 51
5.2. Értékelés 53

Rövidítések jegyzéke

BMP	Bittérképes képállomány (Bitmap)
LSB	Legjelentéktelenebb bit (Least Significant Bit)
RGB	Piros, zöld, kék (Red, Green, Blue)
I/O	Be- és kimenet (Input/Output)
GIMP	GNU Képmanipuláló Program (GNU Image Manipulator Program)
DB	Adatbázis (Database)
SQL	Strukturált lekérdezőnyelv (Structured Query Language)
HSQLDB	Hiper-SQL Adatbázis (Hyper Structured Query Language Database)
API	Alkalmazásprogramozói felület (Application Programming Interface)
JDBC	Java adatbáziskapcsolat (Java Database Connectivity)
AWT	Absztrakt ablakozó eszközkészlet (Abstract Windowing Toolkit)
FUSE	Állományrendszer a felhasználói térben (Filesystem in Userspace)

Ábrák jegyzéke

1.1. A szteganográfia alapelve 1
1.2. A szteganográfiai algoritmusokat jellemző mértékek 2
1.3. Az információ védelmének szintjei 4

2.1. Szórt spektrumú adatrejtés 10
2.2. Szórt spektrumban rejtett adat észlelése 11
2.3. A JPEG algoritmus vázlatos működése 13
2.4. Visszhangkódolás 16

3.1. A program funkcionális egységeinek együttműködése 24
3.2. A program grafikus felületének főablaka 27
3.3. Új adatbázis létrehozása 27
3.4. Hordozók kijelölése 28
3.5. A program bemutatása használat közben 28
3.6. A lineáris allokáció bemutatása egy 4 hordozót tartalmazó stegodrive-on . . . 30

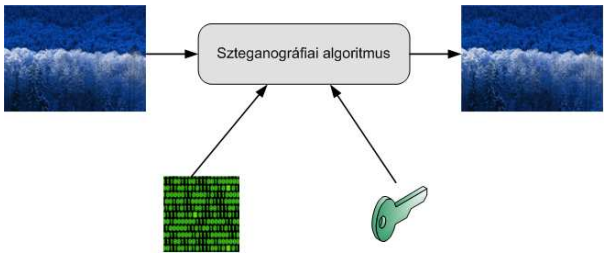
4.1. LSB-változtatás észlelhetősége 36
4.2. Az eredeti fénykép 37
4.3. Az eredeti fénykép jobb felső sarka, nagy kontraszttal 37
4.4. A rejtett adattal teleírt kép jobb felső sarka (0-s küszöb mellett) 37
4.5. A rejtett adattal teleírt kép jobb felső sarka (15-ös küszöb mellett) 38
4.6. Átlagos kapacitás (bájtban) 45

1. fejezet

Bevezető

A *szteganográfia* olyan módszerek összefoglaló neve, melyek egy *hordozó médiumba* elrejtenek valamilyen üzenetet, így kiadva az úgynevezett *sztego médiumot* (1.1. ábra). Az ilyen algoritmusok általában egy kulccsal paraméterezhetőek (*sztego kulcs*) – hasonlóan a kriptográfiához, a szteganográfiában sem szerencsés az algoritmus titkosságára alapozni a biztonságot (lásd még *Kerckhoffs-elvek* [10]).

A hordozó sokféle lehet. Gyakran az a leglényegesebb tulajdonsága, hogy nem kelt gyanút. Pusztán ez alapján a kritérium alapján képek, hangfájlok, ritkábban szövegek, futtatható binárisok, illetve hálózati adatforgalom szoktak hordozók lenni. Az ezeken használható szteganográfiai módszereket a 2. fejezetben fejtem ki.



1.1. ábra. A szteganográfia alapelve

1.1. A szteganográfiával kapcsolatos problémák

A szteganográfiai algoritmusokat többféle mértékkel jellemezhetjük [1]. A fontosabbak a következők:

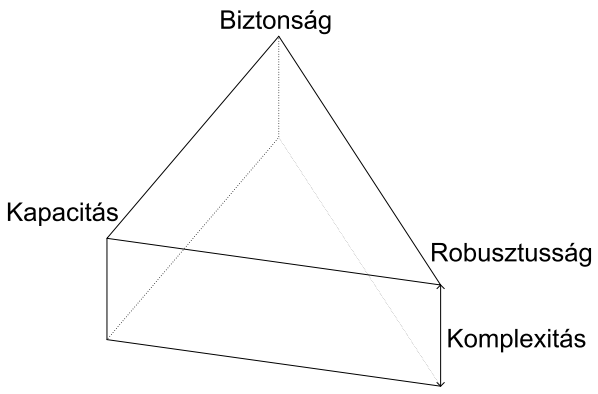
kapacitás: a hordozó hány %-a használható adatrejtésre. Egyes módszereknél ez a hordozó (bitekben értendő) méretétől függ csak, míg másoknál a hordozó egyéb tulajdonságai is befolyásolhatják ezt.

észrevétlenség: a szteganográfia által a hordozóba bevitt transzformáció mennyire árulja el a rejtett információ jelenlétét. Szokás még biztonságnak is nevezni ezt a mértéket. Mindig a szteganográfiai algoritmussal szemben támasztott igényektől függ, hogy milyen modell alapján értelmezzük az észrevétlenséget.

robusztusság: a sztego médium transzformációinak (például kép újratömörítése) mennyire áll ellen a rejtett információ. Ez főleg a vízjelek esetében fontos kritérium, de szteganografikus fájlrendszerek analízisekor is érdekes lehet.

komplexitás: a szteganográfia kivitelezéséhez (az adott algoritmus lefuttatásához) mekkora tár-, illetve időkomplexitás tartozik.

Az első három mértéket szokás egy háromszögként tekinteni. A komplexitással együtt ezt az 1.2. ábrán látható módon lehet például ábrázolni.



1.2. ábra. A szteganográfiai algoritmusokat jellemző mértékek

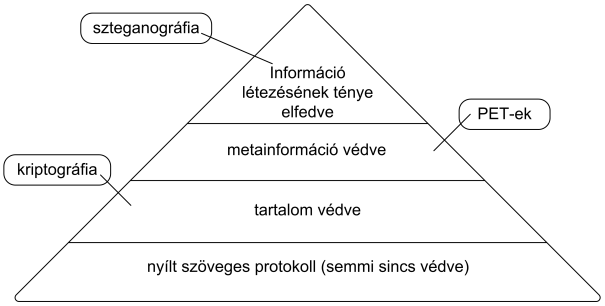
Jelen diplomadolgozat témája egy olyan program megtervezése és implementációja, mely a felhasználó által kiválasztott hordozó állományok szteganografikus kapacitását összefüggő, véletlen hozzáférésű tárterületként láttatja – a koncepciót a továbbiakban *stegodrive* néven illetem. A program ezen felül titkosítással is növeli a biztonságot. Opcionálisan törékeny vízjelekkel hitelesíti az adatot, valamint az adatrejtés előkészítésekor lehetőséget ad a láthatóság és a kapacitás közti kompromisszum szabályozására.

Az észrevétlenség definiálására nem létezik teljesen egységes szemlélet. Kétféle definíciót ismertet [8] és [4]. Az egyik egy információelméleti modell, mely a hordozó entrópiája alapján ad a szteganográfia észrevehetetlenségére mértékeket. Ez jól használható egyrészt olyan helyzetekben, amikor a rejtett adat statisztikailag véletlenszerű (ez a valós helyzetekben praktikusan akkor teljesül, ha a rejtés előtt titkosítás történt), illetve olyan módszereknél, ahol a robusztusság a fő kritérium, nem az észrevétlenség. Egy másik modell probabilisztikusan közelíti meg az észrevétlenség kérdéskörét. Eszerint az észrevétlenséget egy adott szteganalitikai algoritmusra kell definiálni annak helyes pozitív és fals pozitív valószínűségeivel. Ha ezek távolsága minden határon túl leszorítható, akkor nevezzük a szteganográfiai algoritmust az adott szteganalízissel szemben tökéletesen biztonságosnak. Ennek a modellnek előnye a másikkal szemben az, hogy a hordozó statisztikájától független, hátránya viszont, hogy a valószínűségeket minden szteganalitikai módszerhez külön kell definiálni, illetve mérni.

Jelen dolgozatban a biztonságot a statisztikai tulajdonságok megváltozásának mértékével fogom jellemezni [20]. Ennek előnye, hogy viszonylag általános, tehát nagyon sokféle rejtési algoritmusra alkalmazható, illetve, hogy ez a gyakorlatba könnyebben átültethető szemlélet.

1.2. A szteganográfia és a privátszféra kapcsolata

Az 1.3. ábrán az információ védelmének szintjei [18] láthatóak. Az első szint a kriptográfia alkalmazása: ekkor az elrejtendő üzenetet úgy kódoljuk, hogy a tartalomhoz illetéktelen harmadik személy ne férjen hozzá. A második szintet a PET-ek képviselik, melyek a tartalmon kívül az ún. *metaadatokat* is elfedik. (Egy e-mailnél például fontos kívánalom lehet, hogy a



1.3. ábra. Az információ védelmének szintjei

feladó és a levél által bejárt SMTP szerverek listája ne legyen visszakövethető; erre a célra egy *anonim levelezőt* (*anonymous remailer*) használhatunk). A modell legfelső szintjét a szteganográfia alkalmazása képviseli: ilyenkor a cél az, hogy az információ létezése is titokban maradjon. Ha ez nem teljesül, akkor a védelemnek – az alkalmazott módszerek függvényében – maximum második, első, illetve nulladik szintjét valósítjuk meg. Látható, hogy a szteganográfia alkalmazása jól kiegészíti a *privátszférát erősítő technológiákat* (*PET*-eket), illetve a *kriptográfiát*.

Elképzelhető, hogy a ma még kevéssé elterjedt szteganográfia később nagyobb szerepet kap a viszonylag széles körben használt kriptográfiai módszerek mellett [12]. Ennek egyik hajtóereje lehet az uniós adatvédelmi filozófia változása, lazulása [13].

2. fejezet

A szteganográfia irodalma

Az alábbiakban a szteganográfiai algoritmusokat sorolom kategóriákba. Sokféle dimenzióban elvégezhető hasonló kategorizálás, ezekből hármat emelnék ki: a tartományt, az adaptivitást, illetve a célt.

2.1. Csoportosítás tartomány szerint

A hordozó médiumon kétféle tartományban végezhetőek módosítások [7]: a *tértartományban*, illetve, valamilyen *transzformációs tartományban*. Előbbi a hordozó „értelmének" manipulálását jelenti, vagyis képnél a pixelekét, hangnál a hangmintákét, szövegnél a betűkét stb. Az ilyen módszerek viszonylag egyszerűek és intuitívak, viszont gyakran statisztikai anomáliákat okoznak, illetve rossz adatrejtési hely választása esetén emberi érzékekkel is érzékelhető lehet a változás.

A transzformációs tartományban történő adatrejtéshez a hordozót valamilyen transzformáció után manipuláljuk. Ilyen lehet például JPEG képek esetén a *diszkrét koszinusz transzformáció* (*DCT*) [9]. A DCT alkalmazása után kapott együtthatókon végzett módosítások ugyanis jobban „szétkenődnek" a tértartományban – elképzelhető, hogy a szteganalízis során ezt egyszerű kvantálási zajnak tudják be. Az ilyen módszerek viszonylag bonyolultak, és alapos ismereteket tételeznek fel az adott transzformáció működéséről, valamint a manipuláció és a tértartománybeli adathalmaz kapcsolatáról.

Az a megállapítás is érdekes lehet, hogy egy tértartományban nagyon bonyolultnak tűnő transzformáció egy alkalmasan megválasztott transzformációs tartományban egyszerűen leírható, és a vele megvalósított szteganográfia ebben a tartományban támadható. Például a 2.6.1. szakaszban leírt szórt spektrumú adatrejtésben is szerepet játszanak transzformációs tartománybeli megfontolások, hiába történik az adat elrejtése tisztán a tértartományban.

2.2. Csoportosítás adaptivitás szerint

A szteganográfia által végzett manipulációk történhetnek *adaptívan*, illetve *nemadaptívan*. Az adaptivitás lényege, hogy valamilyen módszerrel megpróbáljuk az adatrejtést optimalizálni az emberi érzékszervek, illetve a szteganalitikai szoftverek ellen. A nemadaptív módszerek egyszerűek, viszont könnyen lehet, hogy a módosítások egyes hordozók esetén felismerhetőek szemmel-füllel vagy statisztikai elemzéssel. Az adaptív módszerek hátránya, hogy bonyolultak: nehéz programmal leírni, hogy milyen terület alkalmas az adatrejtésre. Ellenben így elérhető lehet a hordozó statisztikai tulajdonságainak megtartása, illetve kihasználható az emberi érzékszervek tehetetlensége is.

2.3. Csoportosítás cél szerint

A szteganográfiai módszerek alapvetően három csoportba sorolhatóak: *egyszerű adatrejtés*, *vízjelezés*, illetve *rejtett csatorna*. Az első kategóriában a cél a hordozótól független adat elrejtése – az algoritmus szempontjából nem számít az elrejtett adat formátuma, jellege, célja stb.

A vízjelezés kategóriájába olyan módszerek tartoznak, melyek a hordozó megjelölését szolgálják. Ebben a csoportban törékeny és robusztus vízjeleket különböztetünk meg. Előbbi célja a sztego médium hitelesítése (a kriptográfiai digitális aláírással analóg módon), oly módon, hogy a vízjel a sztego médium lehetőleg minél kisebb transzformációja esetén (pl. JPEG újrakódolás) megsérül. A robusztus vízjel lényege, hogy a rejtett adat csak a sztego médium

elfogadhatatlan mérvű minőségromlást okozó transzformációja (pl. hang mintavételezési frekvenciájának csökkentése 44 kHz-ről 8 kHz-re) esetén tüntethető el.

Megjegyzendő, hogy léteznek olyan programok, melyek a robusztus vízjelek minél kisebb minőségvesztéssel történő eltávolítására szakosodtak. Ilyen például a *StirMark*[1] nyílt forráskódú szoftver, melyet azzal a céllal írtak meg, hogy normaként szolgáljon egy robusztus vízjel erősségének mérésére. A program többféle transzformációt (például D/A/D átalakítás utánzása, zajosítás, geometriai transzformációk) is megpróbál a sztego médiumon, és a vízjelet felismerő algoritmust „megkérdezve" dönt arról, hogy a vízjel eltüntetése sikeres volt-e.

A rejtett csatornák alá olyan módszerek tartoznak, melyek speciálisan megkonstruált kriptogramokban, illetve kriptográfiai kulcsokban végeznek adatrejtést. A cél gyakran kleptográfiai jellegű, például a privát kulcs néhány bitjének kiszivárogtatása a digitális aláírásban. Rejtett csatornákkal kis kapacitásuk és jelen dolgozattal kapcsolatos irrelevanciájuk miatt a továbbiakban nem foglalkozom.

2.4. Szteganalízis

Ahogy a kriptográfiának a kriptanalízis, úgy a szteganográfiának a *szteganalízis* tekinthető „ellenpontjának". Fontos megállapítás, hogy a szteganalízis célja annak eldöntése, hogy van-e rejtett üzenet egy médiumban vagy nincs – az, hogy mi a rejtett adat, általában nem fontos, hiszen pont a szteganográfia célja, a rejtett információ létezésének titokban maradása bukott el. Nem célszerű tehát arra koncentrálnunk, hogy megtudjuk, mi a rejtett üzenet, elég, ha viszonylag nagy bizonyossággal el tudjuk dönteni, hogy a vizsgált médium tartalmaz-e egy konkrét szteganográfiai algoritmus által rejtett adatot vagy sem.

A szteganalízis sokféle módszert alkalmaz. Gyakori ezek közül a statisztikai elemzés, mely akkor használható, ha az adott szteganográfiai algoritmus alkalmazása bizonyos statisztikai anomáliákból kimutatható. Ilyen statisztikatorzulási vonásokat hívatott felismerni az 1.1. fejezetben már említett elsőrendű statisztikai vizsgálat. Szofisztikáltabb statisztikai módszerek

[1] http://www.petitcolas.net/fabien/watermarking/stirmark/

[21] a vizsgált valószínűségi változók magasabb momentumait is elemzik, így a csak az elsőrendű statisztikát korrigáló algoritmusok ezekkel is kiszűrhetőek.

Egy másik módszer lehet egyes szteganográfiai algoritmusok esetén a nyers erő alapú, illetve szótáras támadás – ilyenkor azt a jelszót keressük, melyből a sztego kulcs származtatásra került (szótáras támadás esetén azzal a feltevéssel élünk, hogy a jelszó egy könnyen kitalálható értelmes szó).

Valamelyest kilógnak a sorból a heurisztikus módszerek, melyek nem algoritmushoz kötöttek – épp ellenkezőleg, az a céljuk, hogy ismeretlen szteganográfiai algoritmusokat is detektálni tudjanak. Ehhez két adatbázist kell fenntartani: egyet „üres" hordozókból, és egy másikat rejtett információt tartalmazó sztego médiumokból. Az algoritmus ezen mintákban próbál szabályosságokat felismerni, és bizonyos valószínűséggel képes következtetni eddig ismeretlen szteganográfia jelenlétére is[2].

A következőkben a szteganográfia szakirodalmában fellelhető rejtési algoritmusokat veszem szemügyre, a hordozó típusa szerint csoportosítva. Bemutatok továbbá olyan algoritmusokat is, melyekkel – feltehetőleg egyszerűségük okán – nem publikáció, hanem például proof-of-concept programkód formájában találkoztam.

2.5. Adatrejtés szövegekbe

A szövegek tartoznak talán a legkevésbé hasznosítható hordozók közé, tekintve, hogy kevesebb ártatlannak tűnő módosítás képzelhető el bennük már pusztán az ábécé meglehetősen véges értékkészlete miatt. Továbbá a szövegek meglehetősen kis kapacitást képviselnek, amit már pusztán egy átlagos fájl hosszát tekintve is beláthatunk.

A módszerek többnyire az emberi szem elől történő elrejtésre koncentrálnak. Erre példa a CR-LF karakterek manipulálása, valamint a sorok végére extra szóközök beillesztése. Inkább a szteganalitikai szoftverek elleni védettséget célozza a vesszőhasználat variálása, mely – tekintettel például a helyesírás-ellenőrző programok mai állapotára – nehezebben felismer-

[2]Proof-of-concept implementáció: http://www.outguess.org/detection.php

hető számítógéppel. Például lehetséges minden „és” szó után vesszőt tenni, ha 1-est akarunk kódolni, és törölni azt, ha 0-t[3].

Összességében a szövegek a kapacitás és az észrevétlenség oldaláról is szenvednek hiányosságoktól, ezért a továbbiakban nem foglalkozom velük.

2.6. Adatrejtés képekbe

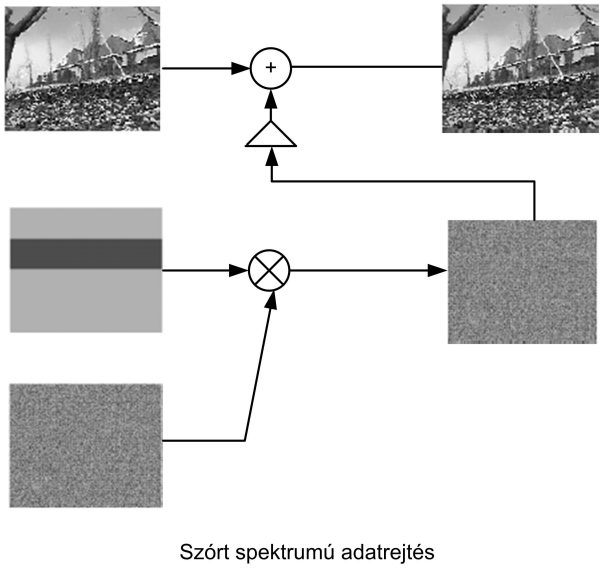
A képek talán a legelterjedtebb hordozók a szteganográfiában, köszönhetően általános jelenlétüknek az Interneten, illetve az e-mailt használók levelesládáiban. A képekbe történő adatrejtésnél meg kell különböztetnünk a tömörítetlen és a tömörített hordozók eseteit.

2.6.1. Adatrejtés raszteres tömörítetlen képekbe

Gyakoran használt algoritmus a *legjelentéktelenebb bitbe* (*least significant bit*, *LSB*) történő adatrejtés. A módszer mindössze annyiból áll, hogy a kép minden egyes pixelén minden színkomponens LSB-jét rendre az elrejtendő adat bitjeire állítjuk, abban bízva, hogy a változás nem lesz emberi szemmel felismerhető. Könnyen belátható, hogy az eljárással egy *vörös-zöld-kék* (*RGB*) *komponensekkel* leírt képbe $pixelek \cdot 3$ darab bit rejthető el [3]. A módszer hátránya, hogy a kép homogén részeiben esetleg emberi szemmel is észrevehető zajok jelentkezhetnek. Ezek még jelentősebbek lehetnek akkor, ha a *színsíkokat* (*colour plane*) külön-külön vizsgáljuk. Mindezeken felül az elrejtett adat megsérül a sztego médium gyakorlatilag bármilyen veszteséges tömörítése, illetve geometriai transzformációja esetén.

Az LSB módszer az emberi érzékszervek számára észrevehetetlenebbé tehető, ha adaptivitást viszünk bele. Egy ilyen eljárás, ha a képpontok között definiálunk egy normát, és csak egy bizonyos küszöbérték alatt használjuk fel a vizsgált pixelt, egyébként átugorjuk. Ilyen norma lehet például a pixel környezetében levő további képpontok valamilyen statisztikai tulajdonsága, például szórása. A szórásból lehet következtetni arra, hogy az adott képrészlet mennyire heterogén, és a túl homogén részek kihagyhatóak az adatrejtésből. Ennek a módszernek az a

[3] A vesszőhasználat esetlegessége persze – ismerve a mai helyesírási kultúrát – valószínűleg az olvasók többségének sem fog feltűnni.



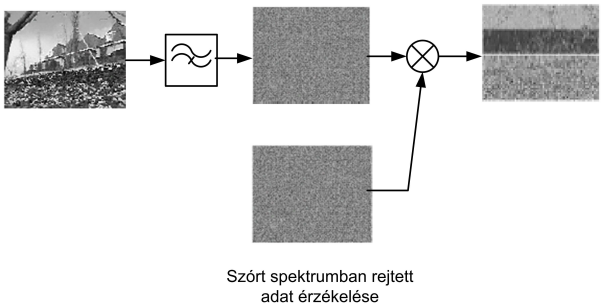
2.1. ábra. Szórt spektrumú adatrejtés

hátránya, hogy az adatrejtésre alkalmasnak ítélt képrészletbe rejtett adat által bevitt változással már nem teljesül a norma – ezzel az eshetőséggel feltétlenül számolni kell a rejtett adat visszanyerésekor. Belátható továbbá, hogy a szabad szteganográfiai kapacitás becslése sem triviális ennél a módszernél. Mindez azonban kiküszöbölhető, ha az algoritmus nem számítja be a rejtéshez használt pixelt a szórásba, csak a környezetét. Ezt az algoritmust a 4.1. fejezetben fogom elemezni.

További javítás lehet például, ha „kulcsosítjuk” az algoritmust, és egy álvéletlen permutáció alapján keressük meg az adatrejtésre használható helyeket. Ezen felül érdemes lehet az elrejtendő adatot *Gauss-féle zajjal* (*Gaussian noise*) modulálni, mert így a hatás egy digitális fényképezőgép *töltéscsatolt eszköze* (*charge-coupled device*, *CCD*) által bevitt hibára fog utalni [16]. A Gauss-féle zaj egy normális eloszlású valószínűségi változó által felvett értékek összessége, mely viszonylag könnyen utánozható egy álvéletlen generátor segítségével[4].

Raszteres képbe történő adatrejtésre használható továbbá a *szórt spektrumú adatrejtés* (lásd a 2.1. és a 2.2. ábrát [11]). Az eljárás első lépése, hogy az elrejtendő adatokban a 0-kat -1-ekkel cseréljük ki, majd *kiterjesztjük*, vagyis az elrejtendő adatot többször egymás után írjuk. Az így kapott bitsorozatot ezután egy *álvéletlen zajjal* moduláljuk – a zajgenerátor magja lesz

[4]http://www.dspguru.com/howto/tech/wgn.htm



2.2. ábra. Szórt spektrumban rejtett adat észlelése

a sztego kulcs. A moduláció eredményét egy amplitúdóval megszorozzuk, majd hozzáadjuk az eredeti képhez.

A szórt spektrummal elrejtett adat észleléséhez (2.2. ábra) a sztego médiumot egy *felüláteresztő szűrőn* eresztjük át. Ekkor a zaj egy közelítését kapjuk vissza, még hozzá azért, mert míg a hordozó médiumban általában a kisfrekvenciás komponensek a dominánsak, addig a zajban a nagyfrekvenciásak. A szűrés eredményét az álvéletlen zajjal demodulálni kell, és el kell osztani az amplitúdóval. Az így kapott eredményben megfelelő kiterjesztés, amplitúdó és szűrési frekvencia esetén dominánsak lesznek a kiterjesztett adathoz tartozó bitek. Az amplitúdó továbbá kompromisszumot jelent a minőségromlás és a kiszűrhetőség között, míg a kiterjesztés mértéke a kapacitás és a kiszűrhetőség közti egyensúlyt képes beállítani.

A szórt spektrum módszerének leggyakoribb felhasználási területe a robusztus vízjel. Az elrejtett adat ugyanis ellenálló a zajosítással szemben, ha az amplitúdó megfelelően van meghatározva, illetve a kiterjesztés jellege miatt például a képből való levágás ellen is védett. További érdekes felhasználási terület a nyilvános kulcsú vízjelezés. Ekkor két, speciális zajgenerátor segítségével két darab szórt spektrumú vízjelet helyezünk el a képen, és az egyik magot nyilvánosságra hozzuk. A nyilvános maggal lehet ellenőrizni a nyilvános vízjel meglétét, ellenben nem lehet vele eltávolítani a titkos vízjelet.

Törékeny vízjelezési megoldás a [19] forrásban leírt módszer[5]. A képet először $n \times n$-es blokkokra osztja, majd egy speciális, $n \times n$-es mátrixot készít, melyben egy elem 0, ha az eredeti blokkban ugyanazokon a koordinátákon a blokk átlagértékénél kisebb képpont van,

[5]Ez egy szürkeskálás képekre megkonstruált algoritmus, mindazonáltal a szerzők szerint színes kép kék színsíkján is hasonló biztonságot szavatol.

máskülönben 1. A cél a blokkot úgy alakítani, hogy a mátrixra teljesüljön, hogy az elemek összege plusz egy álvéletlen bit modulo 2 a vízjel kódolandó bitjét adja ki. A módosítás után egy hibaszétterjesztési eljárással biztosítják, hogy a statisztikai tulajdonságok ne térjenek el nagyon a blokkon belül. A vízjel egy bitjének kinyerése ezek után majdhogynem triviális: csak a blokkokhoz tartozó mátrixokat kell származtatni, valamint összeadni az elemek összegét az álvéletlen bittel. Látható, hogy az algoritmus rendelkezik egy igen előnyös tulajdonsággal: megmondható, hogy melyik vízjelbiteken történt a beavatkozás.

2.6.2. Adatrejtés palettás tömörítetlen képekbe

A *palettás képeknél* más jellegű módszerek használata szükséges. Az ilyen képekben a használható színek halmaza darabonként van definiálva az általában a fejlécben elhelyezkedő palettában, és minden képponthoz az az index van beírva, ahol a palettában a megjeleníteni kívánt szín elhelyezkedik. Ha az index LSB-jét megváltoztatjuk egy pixelnél, könnyen megtörténhet, hogy az eredetitől teljesen eltérő színt látunk majd az adott képponton, vagyis a „naiv" módszer itt nem működik. A palettás képekre is többféle adatrejtési megközelítés alkalmazható.

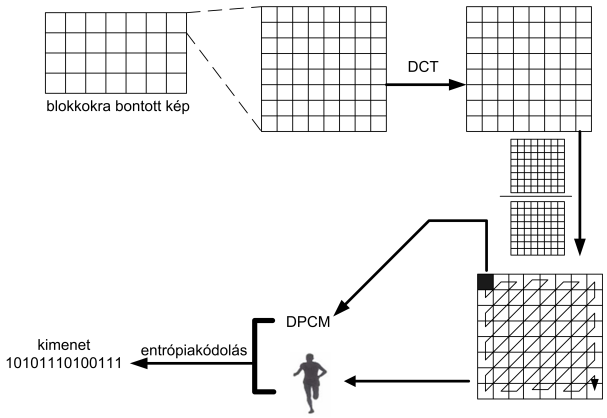
A legegyszerűbb a palettába történő adatrejtés, vagyis amikor a paletta permutációja rejti az adatot. Ennek az algoritmusnak nyilvánvaló hibája a kis kapacitás, illetve a függetlenség a kép dimenzióitól.

Az egyszerű LSB algoritmusra eredményében hasonlító módszer a közeli színekre való váltás[6]. Ehhez a palettát kiterjesztjük az eredetiekhez hasonló, de nem azonos színekkel. Az új színeket úgy határozzuk meg, hogy azok kék komponensében egy 0-s LSB 0-t, 1-es LSB 1-et kódol. Egy álvéletlen permutációval szétszórhatjuk a rejtett adat bitjeit. Az algoritmus hatására a kép némlieg zajosabbá válik, de ez például egy fénykép esetében nem kelt gyanút, tekintve, hogy a palettás formátumban mentett fotók gyakran egyébként is „zajosak" a kevés szín miatt.

[6]Proof-of-concept kód itt: http://www.codeproject.com/KB/security/steganodotnet11.aspx

2.6.3. Adatrejtés tömörített képekbe

A tömörített képeken többnyire a transzformációs tartományban érdemes adatrejtést végrehajtani, mert a tömörítés valószínűleg tönkretenné a képtartományban elrejtett adatot. A *JPEG algoritmus* egy elterjedt képtömörítési módszer, melyhez sokféle szteganográfiai megoldás született már. Az algoritmus vázlatos működése a 2.3. ábrán látható [5]. A képet először 8-szor 8 képpontból álló blokkokra kell bontani, azokat át kell vinni a diszkrét koszinusz transzformációval a DCT-együtthatók tartományába. Az eredménymátrix bal felső sarkában található az ún. *egyenáramú (DC) komponens*, míg a többi elemet *váltakozó áramú (AC) komponenseknek* nevezzük. A 8-szor 8-as DCT-mátrixot elemenként el kell osztani egy másik, ugyanilyen dimenziójú mátrix elemeivel (ez az ún. *kvantálás*), és matematikai kerekítést kell végezni. Az eredménymátrix elemeiből az AC együtthatókat cikkcakkban haladva *futamhosszkódolással (Run Length Encoding, RLE)* tömörítjük, míg a DC komponenst *különbségi impulzuskód-modulációval (Differential Pulse Code Modulation, DPCM)* kódoljuk. Legvégül egy veszteségmentes tömörítési lépés adja a kimenetet.



2.3. ábra. A JPEG algoritmus vázlatos működése

Az egyik JPEG-hez használható szteganográfiai algoritmus az *együttható-kerekítéses módszer*[7]. Ez az eljárás a kvantálási lépést módosítja: kerekítés előtt megkeresi azokat az együtthatókat, amelyek törtrésze közel van a $0{,}5$-hez, és ezeket a megfelelővel ellentétes irányban (1-es), illetve normál módon (0-s) kerekíti, abban bízván, hogy ez az apró módosítás nem

[7] http://qosip.tmit.bme.hu/twiki/pub/Main/InfoSzolgBizt/11-Adatrejtes_kepekben.pdf

lesz feltűnő a képtartományban. A módszer hátránya, hogy az üzenet visszanyeréséhez szükség van az eredeti hordozó médiumra – máskülönben nem tudjuk, hogy melyik kerekítéseket „rontottuk el”.

Egy másik JPEG adatrejtési módszer a (lehetőleg kis jelentőségű, vagyis a táblázat jobb alsó sarkához közeli) DCT együtthatókban történő LSB kódolás. A kerekítéses algoritmussal ellentétben itt nincs szükség az eredeti hordozóra az üzenet kinyeréséhez. Ahogy azt már az előzőekben láttuk, itt is célszerű sztego kulcstól függő álvéletlen permutációt használni az adatrejtéshez. Figyelni kell ezen felül arra, hogy a cikkcakk azonos értékű futamainak „közepébe” ne ékeljünk idegen értéket, mert az gyanút kelthet – normális esetben ez a jelenség nem szokott fellépni kis jelentőségű együtthatóknál.

Végül az OutGuess[8] nevű algoritmusról is érdemes említést tenni. Ez a JSteg algoritmuson alapszik, mely egy speciális LSB kódolás a diszkrét koszinusz transzformációs tartományban. A JSteg azonban statisztikai elemzéssel könnyen detektálható „párosodást” idéz elő az AC együtthatókban [22], ezért az OutGuess a JSteg futása során nem kihasznált AC együtthatókat úgy módosítja, hogy az elsőrendű statisztika helyrebillenjen.

2.7. Adatrejtés tömörítetlen hangfájlba

A tömörítetlen képekhez hasonlóan itt is a lehető legegyszerűbb adatrejtési séma a hangtartománybeli LSB kódolás [3]. A hanghullámot képező minták legjelentéktelenebb bitjeit az elrejtendő üzenetnek megfelelően átbillentve itt is lehetséges adatrejtés. A módszer hátránya továbbra is az, hogy a csendesebb részeknél az emberi fül meghallhatja az üzenet által bevitt zajt. Adaptivitást visz az egyszerű LSB kódolásba a paritásos módszer. Ilyenkor egy álvéletlen generátor alapján LSB-csoportokat választunk, és egy előre definiált norma alapján keressük meg a csoport megváltoztatandó bitjét. Az elrejtett információt ezután a csoport paritása kódolja. Mindkét algoritmus hatalmas hátránya, hogy nem robusztusak a tömörítéssel szemben [14].

[8] http://www.outguess.org/

Az előbbi két algoritmusnál szofisztikáltabb, transzformációs tartománybeli a *fáziskódolás* – komoly előnye, hogy nem visz be zajt a hordozóba [14]. Az algoritmus az üzenet hosszával megegyező szegmensekre bontja a hangot, majd a mintákon *diszkrét Fourier-transzformációt (DFT)* alkalmaz. A legelső szegmensben a fázisszögeket $90°$-ra, illetve $-90°$-ra állítja be a 0, illetve az 1 kódolásához. A további szegmensek fázisszögeit ezután úgy kell módosítani, hogy a szegmensek közti eredeti relatív fáziskülönbség megmaradjon – máskülönben füllel detektálható torzulás következne be a hangban. Legvégül inverz diszkrét Fourier-transzformációval visszatérünk a hangtartományba, és a szegmenseket konkatenálva megkapjuk a kimenetet[9]. A módszer legnagyobb hátránya a kicsi kapacitás: csak az első szegmens hordozhat információt, viszont annak hossza nem növelhető minden határon túl, mert a frekvenciakomponensek közti fáziskülönbségek egyre markánsabban kiütköznek.

Az emberi fül tehetetlenségét kihasználó, hangtartománybeli módszer a *visszhangkódolás* [14], mely az *elfedési jelenségen* alapul, nevezetesen, hogy egy erősebb hangimpulzus egy ideig képtelenné teszi a fület gyengébb hangimpulzusok észlelésére (2.4. ábra). Ha tehát egy erős impulzust kis időn belül kisebb energiával megismétlünk, akkor az emberi érzékszervek számára a változás nem detektálható. A beiktatott és az eredeti impulzus időbeli távolsága kódolja az elrejtett bit értékét. A visszhangkódolás hátránya, hogy az MP3-ba történő tömörítéssel szemben nem robusztus, tekintve, hogy ez a tömörítés nagy mértékben épít az elfedett hangminták kiiktatására.

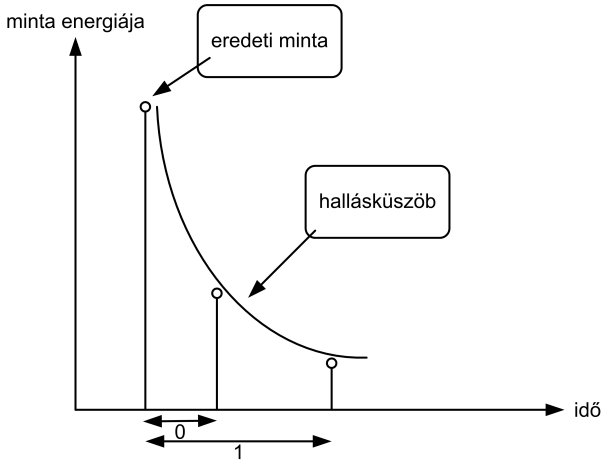
2.8. Adatrejtés videofolyamba

A videó nem más, mint egymás mellé tett hang- és képinformáció. Intuitív tehát az a következtetés, hogy a hangoknál és a képeknél alkalmazott módszerek (LSB, visszhang stb.) ugyanúgy alkalmazhatóak maradnak[10]. Valóban, egy tömörítetlen videojel raszteres képek sorozatának tekinthető, és méreténél fogva az adattárolási kapacitása hatalmas[11]. Fontos azon-

[9] Az üzenet kinyeréséhez értelemszerűen a ismerni kell a szegmenshosszt.

[10] Proof-of-concept kód itt: http://www.codeproject.com/KB/security/steganodotnet4.aspx

[11] Mindazonáltal a tömörített videofolyamokban is lehetséges adatrejtés; egy Motion JPEG kódolással vagy csak B-képekből álló MPEG-videó pedig diszkrétkoszinusztranszformált képek sorozatával analóg.



2.4. ábra. Visszhangkódolás

ban felismerni, hogy a videofolyam képei statisztikai-dinamikai szempontból egyáltalán nem függetlenek egymástól, és ez a függőség szteganográfiai szempontból kiaknázható.

Egy érdekes megoldást mutat be [2]. Itt a szteganográfia a videó hibajavító kódolására kerül felhasználásra: egy kis zaj bevitele a képekbe sokat javít a jelátviteli szinkron elvesztése után a hiányzó képi információ megtalálására[12].

Egyszerű adatrejtésre, illetve vízjelezésre használható a StegoVideo[13], melynek egyik fő előnye, hogy robusztus a jel tömörítésével szemben. (A tömörített jelbe történő rejtés nagyon fontos tulajdonság. Tömörítetlen videókkal legfeljebb a videoszerkesztéssel foglalkozó szakemberek rendelkeznek nagy mennyiségben.) A robusztusság eléréséhez konvolúciós hibajavító kódolási technikákat, illetve Viterbi-dekódolást [15] alkalmaznak.

2.9. Adatrejtés binárisba

Futtatható állományokba is lehetséges adatot rejteni, még hozzá úgy, hogy a program funkcionalitása nem változik. Ehhez gépi kódú utasítások egy halmazát le kell képezni egy másik, ugyanolyan funkcionalitású utasításhalmazra, illetve lehetséges olyan veremkezelő utasítások beszúrása is, melyek végrehajtása után a verem végülis érintetlen marad. Bármelyik

[12] A szerző saját tapasztalata, hogy egy ilyen megoldásnak volna helye a mai magyar digitális televíziózásban is.

[13] http://compression.ru/video/stego_video/index_en.html

megközelítést is választjuk, a módszernek megmarad az a hátránya, hogy assembler-, illetve processzorarchitektúra-függő lesz[14].

2.10. Szteganografikus fájlrendszerek

Egy klasszikus értelemben vett szteganografikus fájlrendszer a rejtett adattárolásnak egy meglehetősen kényelmes módja. Általában megvalósítja a fájl-könyvtár szemantikát, és az operációs rendszer számára egy illesztőprogramon keresztül valóban fájlrendszernek látszik. Az alábbiakban bemutatom a jelentősebb implementációkat, koncentrálva azok szteganografikus biztonságára. Az egyéb paramétereket a 2.10.4. fejezetben vizsgálom.

2.10.1. StegFS

A régebbi implementációk közé tartozik a StegFS [23]. Ez egy 2.2-es Linux rendszermagverziók alá írt illesztőprogram, mely egy valódi *ext2* állományrendszer szabad területét használja hordozónak, és az ext2 szemantikáját (pl. jogosultságok) majdnem teljes egészében reprodukálja. A program 15 biztonsági szintet definiál egy-egy kulccsal, melyek egymásba ágyazott szteganografikus „tárolórekeszeket" alkotnak, vagyis egy adott biztonsági szint kulcsának ismerete esetén látjuk az alacsonyabb biztonsági szintek tartalmát, a felsőbb szinten elhelyezett információ viszont védve van. Ez a mechanizmus *hihető letagadhatóságot* biztosít egy támadóval szemben, aki kényszerítheti a felhasználót a sztegokulcs kiadására. Ha egy alsó biztonsági szinthez tartozó kulcsot kap a támadó, se pró, se kontra nem tud következtetést levonni arra nézve, hogy a felsőbb szinteken van-e elrejtett információ. A StegFS ezen kívül sokféle módszerrel (pl. véletlenszerű adatokkal történő felülírás) igyekszik szavatolni a biztonságot olyan támadókkal szemben is, akik időnként „lefényképezhetik" a hordozó fájlrendszer tartalmát.

[14]Egyszerű adatrejtést és törékeny vízjelet használó proof-of-concept implementáció itt: http://crazyboy.com/hydan/

2.10.2. vstegfs

Új, jelen pillanatban is fejlesztés alatt álló implementáció a vstegfs[15], mely a FUSE[16] könyvtárra épülve valósít meg egy szteganografikus fájlrendszert. A StegFS-hez hasonlóan a vstegfs is képes egy másik fájlrendszer szabad területét felhasználni hordozónak, de lehetséges egy fájlba írni az egész állományrendszert.

2.10.3. TrueCrypt Hidden Volume

Szteganografikus fájlrendszernek tekinthetjük *TrueCrypt* nevű kriptográfiai szoftver *rejtett kötet (Hidden Volume)* nevű funkcióját is. A szolgáltatás lényege, hogy egy titkosított meghajtón belül létrehozunk egy másikat – a külső meghajtón ártatlan, ámde titkolni kívántnak tűnő fájlokat tárolunk, míg a belső az igazi titkokat rejti (ide akár egy külön operációs rendszert is telepíthetünk). A hordozó és a rejtett állományrendszer más-más kulccsal fejthető vissza. A rejtjelezési algoritmusok jellege miatt a belső meghajtó véletlenszerű adatnak fog tűnni, és megkülönböztethetetlen lesz a külsőn levő szabad helytől, melyet a TrueCrypt a meghajtó létrehozásakor véletlenszerű adatokkal tölt fel. Mindennek eredményeképp statisztikai elemzéssel nem található meg a belső meghajtó. Egy támadó nem tudja bizonyítani a belső meghajtó létezését a hordozó fájlrendszerhez tartozó kulcs birtokában, következésképp nem szerez tudomást a rejtve tárolt esetleges személyes, illetve üzleti titkokról.

2.10.4. A szteganografikus fájlrendszerek hatékonysága

Biztonság szempontjából a felsorolt implementációk mindegyike közel azonos elvekre épül, így valószínűleg igazán számottevő eltérés nincs ebben a paraméterben. Azt, hogy ezek a mértékek hogyan alakulnak ezeknél az állományrendszereknél, ebben a fejezetben fejtem ki.

Robusztusság szempontjából mindegyik implementációnak lényegében ugyanaz a hiányossága: egy hordozó fájlrendszer szabad területére rejtenek adatot. Ebből következően az operációs rendszer bármikor felülírhatja a rejtett adatokat, ha írás történik a hordozóra – fel-

[15] https://albinoloverats.net/vstegfs
[16] http://fuse.sourceforge.net

használó tipikusan nem képes megszabni, hogy fizikailag hova írja fel az állományrendszer-illesztőprogram a kért adatot. Nyilvánvaló, hogy a jelenség ellen védekezni kell, mert az adatok integritását valamilyen mértékben meg kell óvni.

A TrueCrypt a *belső kötet védelme (Hidden Volume protection)* funkciót biztosítja erre a célra[17]. Ha a felhasználó a külső állományrendszert felcsatolja, az arra szolgáló párbeszéd-panelen megadhatja a belső kötet kulcsát is. Ha a kulcsok helyesek, akkor a TrueCrypt foglaltként tartja nyilván a belső fájlrendszer által elfoglalt helyet, amíg a külső meghajtó fel van csatolva.

A StegFS *blokkreplikációt* alkalmaz az integritás biztosítására [23]. Ennek lényege, hogy a felhasználó a StegFS állományrendszer létrehozásakor megadhatja, hogy egy blokkot hány példányban írjon bele a hordozóba. Magasabb replikációs faktor mellett nagyobb eséllyel találunk sértetlen példányt egy blokkból a hordozóra történő írás után, ellenben nyilván a kapacitásból a példányszámmal arányban veszítünk. A replikációs faktor beállítása ezáltal egy kompromisszum a kapacitás és a robusztusság között.

A vstegfs *blokkreplikációt* és *hibajavító kódolást* egyaránt alkalmaz. A replikációs faktor a dokumentáció szerint fixen 9, és egy blokkon belül 8 bitből 5 tárol adatot, a maradék 3 a hibajavító kód. Így tehát pontosan kiszámolható, hogy a vstegfs a hordozó kapacitásának $\frac{1}{9}$-én tárol egyedi blokkokat, míg egy blokkon belül a veszteség $3 : 8$.

A vizsgált szteganografikus fájlrendszerekből a StegFS-re és a TrueCryptre állnak rendelkezésemre (idő)komplexitással kapcsolatos tapasztalatok. A StegFS esetében a teljesítmény-degradáció 99%-os is lehet [23], ha nagy replikációs faktorral létrehozott fájlrendszerre írunk. A TrueCrypt esetében a komplexitás a felhasznált kriptográfiai algoritmusok függvénye. A legrosszabb esetben három algoritmus láncából előálló *kaszkád*[18] védi az információt, mely mindennapi tapasztalataim szerint nemritkán 80%-os teljesítménycsökkenést okoz.

[17] http://www.truecrypt.org/docs/hidden-volume-protection
[18] http://www.truecrypt.org/docs/cascades

2.11. Formátum gyengeségek kihasználása szteganográfiai célokra

Az alábbi módszerek a fájlformátumokban adatrejtésre kihasználható „gyengeségek” kiaknázását célozzák[19]. Az itt felsorolt algoritmusok közös előnye az egyszerűség, azonban biztonságosnak egyik sem mondható, tekintve, hogy az adott fájlformátumot ismerő szteganalitikus egy hexaszerkesztővel is észreveheti a szteganográfia jelenlétét. Ezek a módszerek továbbá meglehetősen kis kapacitásúnak mondhatóak.

BMP méthamisítás: ez a módszer a BMP kép fejlécében csökkenti a szélességetvagy magasságot 1 pixellel, a baloldalt vagy alul „kihagyott” sort megváltoztatja. Nagyobb méretű kép esetén a változás nem feltűnő, és egészen sok helyet nyerhetünk vele.

JPEG fejléchiba: a módszer lényege, hogy egyszerű konkatenációval „rejtjük el” az adatot egy JPEG képben. Egyértelmű hátrány, hogy a szteganográfia által bevitt adat száz százalékban a fájl végén koncentrálódik, illetve, hogy a fejléc által mutatott méret nincs összhangban a fájl valódi méretével[20].

HTML elemek kódolása: egy HTML dokumentum esetében kihasználhatjuk a böngészők eltérését a szabványoktól – jelesül, hogy az elemek neveit általában kis- és nagybetűk tetszőleges kombinálásával elfogadják, értelmezik[21]. Könnyen kikövetkeztethető, hogy egy elemnév egyetlen betűjével lehetséges 1 bitnyi információt kódolni. A módszer egyértelmű hibája, hogy „ordít róla” az alkalmazása, amint valaki ránéz a HTML kódra[22].

BMP paletta megváltoztatása: egy 256 színű BMP kép palettáján az ún. alfa-csatornát, mely eredetileg a kép átlátszóságáról hordozna információt, nem értelmezik az implementációk. Egyetlen képben így körülbelül 255 bájt rejthető el, ami általában nem

[19] Az itt közölt algoritmusok egy része a 2008. Hacktivity konferencián – ahol a szerő előadóként szerepelt – került felhasználásra. Egyes módszerek implementációjában a szerző is részt vett.

[20] A módszer primitívnek hangzik, de egészen régi implementációk is léteznek rá, mint például a Camouflage. (Lásd http://camouflage.unfiction.com/)

[21] Ez csak akkor igaz, ha nem várunk el XHTML-kompatibilitást. Egy XHTML-ellenőrző kizárólag kisbetűkkel fogadja el az elemneveket.

[22] Esetleg lehetséges egy elem egész nevét azonos betűnagyságúra kódolni, hogy az algoritmus kevésbé keltsen feltűnést. Ilyenkor értelemszerűen nagyon sokat veszítünk a felhasználható kapacitásból.

elégséges, valamint felmerülhet a hordozók csekély száma is – ma már sokkal gyakoribbak a 24 bites képek, melyekben nincsen paletta.

PCX futamhosszkódolás megváltoztatása: a PCX formátumú képekben futamhosszkódolást alkalmaznak. Ha a futamot felbontjuk több darabra, melyek hosszainak összege megegyezik az eredeti futam hosszával, akkor a kép nem változik a tértartományban. Ezáltal lehetséges például a páros hosszú futamokkal 0-t, míg a páratlanakkal 1-et kódolni.

BMP kitöltés: egy BMP képben az egy sorban található pixelek száma 8-cal osztható kell, hogy legyen. Ha ez nem teljesül, akkor minden sorban kitöltést kell alkalmazni, mely képpontok beszúrását jelenti. Ezek a hozzáadott pixelek általában feketék, de ez nem lényeges tulajdonság, mert sosem kerülnek értelmezésre. Következésképpen a kitöltés egésze szteganográfiai célokat szolgálhat.

3. fejezet

A saját szteganográfiai szoftver implementációjának vázlata

A modellezés során kétféle megközelítést használtam fel. Bár a normaértékes LSB adatrejtésnél az emberi érzékelési szempontokat vettem figyelembe, első rendű statisztikai vizsgálatokkal is megvizsgálom a hordozó tartalom torzulását. Ezen kívül megnézem, hogy változnake a sztego médium statisztikai jellemzői, ha az elrejtendő információt előzetesen nem titkosítom.

3.1. Modell a kapacitás-észrevétlenség kompromisszum egydimenziós leképzésére

A kapacitás-észrevétlenség kompromisszumot lehetséges egyetlen számmal (a továbbiakban *küszöbérték*) reprezentálni. Először is minden hordozóhoz megválasztjuk a hozzá tartozó algoritmusokat, és ezeket észrevétlenség szerint sorba rendezzük. (Praktikusan az adaptív algoritmusok lesznek az észrevétlenség felé billenők, míg a nemadaptívak a kapacitás szempontjából jobbak.) A nemadaptív módszer a küszöbérték egy tartományára használatos, míg azon kívül már az adaptív algoritmus érvényesül. Az adaptív algoritmusok mindig rejtenek magukban finomhangolhatósági lehetőségeket, ezért működésük (az adatrejtési egység jósá-

gának megítélése) függővé tehető a küszöbértéktől.

Minden hordozóhoz konstruálhatunk egy igényeinknek megfelelő leképzési függvényt, melybe a nekünk tetsző algoritmusok alkalmazására adunk meg valamilyen normát. A raszteres képek esetében például a következő leképzés alkalmazható. Tegyük fel, hogy a csúszka értéke (x) 0-tól 100-ig terjedhet, ahol 0 a maximális kapacitást, 100 pedig a maximális észrevétlenséget jelenti. Tekintsük a 3.1. összefüggést. Látható, hogy a függvény a küszöbértéket és egy második paramétert a logikai értékek halmazára képez le. A függvény IGAZ értéke esetén mentésre alkalmasnak ítéljük az adott rejtési helyet, míg HAMIS értéknél nem.

$$f(x, I) = \begin{cases} \text{IGAZ} & \text{ha } 0 \leq x \leq 20, \\ LSBad(x, I) & \text{egyébként.} \end{cases} \tag{3.1}$$

Itt x a csúszka értéke, míg I egy halmazt jelöl, melyben az x által kiválasztott függvény számára érdekes képpontok vannak. Az első függvény nem veszi figyelembe x értékét, tehát I értéke irreleváns. A második esetben I az adatrejtési képpont 3×3-as környezete. Az $LSBad$ függvény kifejtése a 3.2. összefüggésben található.

$$LSBad(x, I) = \begin{cases} \text{IGAZ} & \text{ha } \sigma I \geq (x - 20) \cdot \frac{5}{4} \\ \text{HAMIS} & \text{egyébként.} \end{cases} \tag{3.2}$$

Itt az x paraméter 20 és 100 közti értelmezési tartományának 0 és 100 közti skálára normált értéke kerül összehasonlításra az I halmaz elemeinek szórásával. A szórás ugyanis matematikailag viszonylag könnyen értelmezhető képet ad a képpont környezetének homogenitásáról: nagy szóráshoz heterogén, kis szóráshoz homogén környezet tartozik. Az x paraméter növekedésével egyre „paranoidabb” adaptivitási normát kapunk[1].

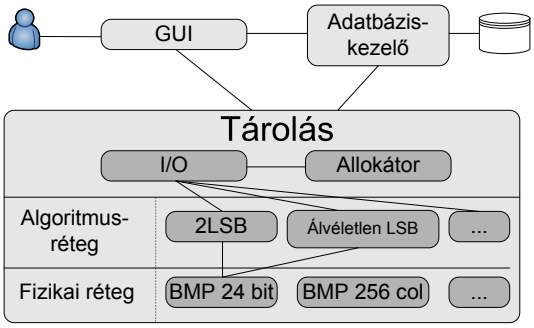
Ha a programban többféle hordozó formátum felhasználását implementáljuk, akkor mindegyikhez külön leképzési függvényre van szükség. A kritérium csupán annyi, hogy 0-tól 100-ig terjedő értékekre történjen a leképzés, a függvény „szakadási helyei” (vagyis azok az érté-

[1] A 100 nem feltétlenül a maximálisan elérhető szórás, de igencsak magas. Ha a legmagasabb szórást is előírhatná a felhasználó normának, akkor alig kapnánk használható képrészleteket. Tapasztalataim szerint körülbelül 40 körül alakul az az érték, ahol még számottevő kapacitást kapunk.

kek, melyek környezetében a felhasznált algoritmus megváltozik) lehetnek formátumonként teljesen különbözőek.

A stegodrive tehát a felhasználó által kiválasztott hordozókba rejt el adatot, a megadott biztonság-kapacitás kritérium szerint. A program célja egy szteganográfiai állományrendszerekhez hasonlatos funkcionalitást biztosítani, habár a jelenlegi implementációban nem támogatott az operációs rendszer számára transzparens működés. A szoftver számára az elrejtett adatok metainformációit, valamint a felhasználandó hordozókat egy külső fájl formájában kell a program rendelkezésére bocsátani, mely az adatbázis elemeit tartalmazó SQL szkript.[2] Adatbázisszervernek a HSQLDB-t [24] használtam, csak operatív memóriában létező adatbázisokkal.

A stegodrive-nak három része van: az adatbáziskezelő, a grafikus felület és a tárolásért felelős rész. A program különböző részeinek együttműködését szemlélteti a 3.1. ábra.



3.1. ábra. A program funkcionális egységeinek együttműködése

[2] Az eredeti elképzelés az volt, hogy a hordozókat láncba szervezzem, és a lánc valamely elemébe elrejtett adatok többek közt a soron következő hordozóra nézve is tartalmaztak volna információt. Az ötlet mellett szól, hogy így egyetlen jelszó szükséges az adatok kinyeréséhez (a titkosítás feloldásához, illetve az esetleges álvéletlen permutációk inicializálásához). Azért döntöttem mégis a külső adatbázisos megközelítés mellett, mert így láncszakadással szemben valamelyest robusztus az implementáció, valamint könnyű redundanciát vinni a tárolásba.

3.2. Az adatbáziskezelő működése

Az adatbázisban négyféle rekord létezik: *hordozo*, *rejtettfajl*, *darab*, *attributum*. A hordozo tábla pusztán a felhasznált hordozókat tartalmazza (új adatbázis létrehozásakor kerül kitöltésre). A rejtettfajl az elrejtett fájlok nevét tartalmazza. A darabok megmutatják, hogy egy adott rejtett fájl mely hordozókban, és azon belül hol (mely ofszeten) került elrejtésre.

A darabok segítenek továbbá a hordozóban levő szabad hely kiszámításában is. Ehhez kellenek a darabokhoz tartozó ofszet, illetve hossz értékek. Az ofszet és a hossz az adott hordozón alkalmazott algoritmus szerinti rejtett bájtban értendő. Fontos, hogy ez a rendszer lehetővé teszi a rejtett fájlok törlését is, egyszerűen a darab-rekordok eltávolítása által[3]. Azonban bizonyos algoritmusoknál az adatrejtésre alkalmas helyek megszűnhetnek, ezért az ofszetek aktualizálására lehet szükség, ha két darab között szeretnénk kitölteni az üres helyet. *Ez azt is jelenti, hogy az érintett algoritmusoknál nem lehetséges a hordozó kapacitásának előre történő meghatározása, mert az a rejtett tartalomtól függ.* Mindazonáltal a jelen implementációban használt algoritmusoknál ez a probléma nem merül fel.

Végül az attributum rekordok a rejtéshez kapcsolódó egyéb információkat tárolják. A jelenlegi implementációban ez a kapacitás-észrevétlenség csúszka értékét, egy salt értéket és a jelszó SHA-1 hash-ét[4] foglalja magában. Utóbbi kettőre egyedül azért van szükség, hogy ellenőrizni lehessen, érvényes-e a megadott jelszó. Ha ugyanis a felhasználó rossz jelszót ad meg, és a sztegokulcs abból kerül származtatásra, akkor a stegodrive-ra történő írás megsértheti a rejtve tárolt adatok integritását (pl. ha a sztegokulcs egy álvéletlen rejtési pozíció sorozat magjaként szolgál).

A program használatához egy adatbázist kell betölteni egy fájlból. Ehhez a következő lépések szükségesek:

1. Adatbázis-kapcsolat megnyitása

[3] A tárolás-törlés ciklusok hosszú távon „elkoptathatják” a hordozót. Ennek az az oka, hogy a szteganográfia által a hordozóba bevitt információveszteséget általában nem lehetséges törléskor kompenzálni. Ez a hatás valamelyest kompenzálható, ha először mindig a törölt elemek helyét töltjük fel újabb rejtett adattal, azonban ez a fajta elhasználódáskiegyenlítés (wear-levelling) bonyolultabb implementációt igényel.

[4] A felhasznált implementáció: http://www.anyexample.com/programming/java/java_simple_class_to_compute_sha_1_hash.xml

2. Adattáblák létrehozása

3. A megadott fájl beolvasása SQL szkriptként.

Az adatbázisréteg fő osztálya a DB, mely a 2. és a 3. műveleteket JDBC Statementek használatával hajtja végre. Ha valamelyik lépés során a JDBC API hibát jelez, egy StegoDBException kivételt kap a hívó a hiba okának körülbelüli leírásával.

Az adatbázisréteghez tartoznak továbbá a Cover, a Stego, a Piece, illetve az Attribute JavaBean objektumok, és szerkezetileg rendre megfeleltethetőek a hordozo, a rejtettfajl, a darab, illetve az attributum táblákban található rekordoknak. Mindegyikhez tartozik a DB osztályban *get* és *remove* metódus, melyek az egyedi azonosító alapján képesek az adatbázisból kikeresni a megfelelő rekordot, illetve törölni azt. Mindkét metódus StegoDBExceptiont dob adatbázishiba esetén, illetve ha a rekord nem létezik. Az adatbázis bővítésére rendelkezésre állnak *add* metódusok, míg a táblák elemeinek megszámolását a *count* kezdetűek végzik.

A DB osztály egy további metódusa képes visszaadni az összes hordozót, valamint egy hordozónevekkel indexelt, Piece tömböket tartalmazó HashMapet. Ezekre a folyamrétegnek van szüksége a rejtés vezérléséhez.

3.3. A grafikus felület

A grafikus felületet a Java AWT API segítségével valósítottam meg. Funkciói a következők:

- Adatbázis létrehozása (a felhasználó megadhatja a hordozókat)
- Adatbázis megnyitása
- Észrevétlenség-kapacitás csúszka
- Fájl elrejtése
- Rejtett fájl kinyerése

- Rejtett fájl törlése
- Jelszó megadása

A program főablakát mutatja a 3.2. ábra. A panel három részre osztható. Baloldalt az adatbázisműveletekkel kapcsolatos gombok találhatóak, középen a rejtéssel kapcsolatosak, jobboldalt pedig a program által visszaadott üzenetek gyűlnek. Az ablak tetején, középen található fehér dobozban kerülnek felsorolásra az elrejtett fájlok. Az itt kijelölt elemre hat a „Rejtett fájl visszahozása” és a „Rejtett fájl törlése” gomb.

A program használatának megkezdéséhez egy adatbázist kell megadnunk, melyből majd dolgozni fog. Ez vagy egy már létező fájl, vagy egy üres, újonnan létrehozandó állomány. Ha az „Adatbázis megnyitása” gombot választjuk, akkor egy hétköznapi „Fájl megnyitása” ablakot kapunk, majd az adatbázisfájl kiválasztása után meg kell adnunk a hozzá tartozó jelszót. Ha rossz jelszót adunk meg, az adatbázis nem kerül megnyitásra.

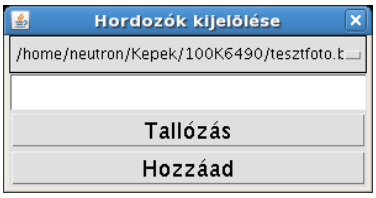
3.2. ábra. A program grafikus felületének főablaka

Ha új adatbázist hoznánk létre, akkor a 3.3. ábrán látható ablakot kapjuk. Itt látható a stegodrive-koncepció egyik pillérét képező kapacitás-biztonság csúszka, melynek pozíciója egy 0-tól 100-ig terjedő értékre képződik le a programban.

3.3. ábra. Új adatbázis létrehozása

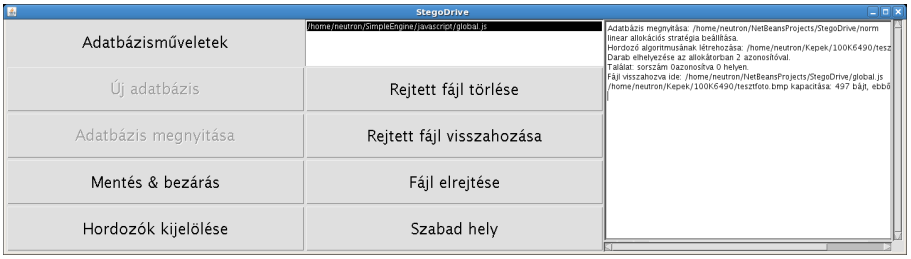
Ha már van nyitott adatbázisunk, a felhasználandó hordozók megadása a következő lépés. A „Hordozók kijelölése” gomb a 3.4. ábrán látható ablakot hozza fel. A felső listában meg-

találhatóak a már kijelölt hordozók. Új hozzáadásához a „Tallózás” gombon való kattintásra feljövő átlagos fájlkiválasztó ablakban választunk egy hordozót, és a „Hozzáad” gombbal hozzáadjuk az adatbázishoz. A jelenlegi implementációban hordozó elvételére nincs lehetőség, ez ugyanis – bár valamilyen szinten hasznos lehet a gyakorlati használat során – rejtett fájlok elvesztését jelenthetné egy már használatban levő stegodrive-on. Az erre adható egyetlen elegáns megoldás az érintett hordozóban levő darabok újraírása a megmaradóakba, azonban ennek implementációja körülményessége és időigénye miatt nem került bele a programba.



3.4. ábra. Hordozók kijelölése

Végül a 3.5. ábra bemutatja a stegodrive-ot használat közben.



3.5. ábra. A program bemutatása használat közben

3.4. Az adatrejtés megvalósítása

A fájlokba rejtendő adatokon először is rejtjelezés történik, melyhez a kulcsot a felhasználó szolgáltatja, egy jelszó formájában[5]. Rejtjelnek az RC4 folyamtitkosító algoritmust használtam, mely könnyen alkalmazkodik bármilyen méretű elrejtendő adathoz, ellentétben egy blokkrejtjelezővel, melynek használatához a kitöltést kell alkalmazni az utolsó blokkon, ha a

[5]Ezt egy SecretKeyFactory segítségével SecretKey-vé alakítom, és a program a továbbiakban ezzel titkosít minden adatot.

fájlméret nem többszöröse a blokkméretnek. A titkosítás opcionális, az a programban kikapcsolható az adatbázis létrehozásakor. A rejtjelezés kikapcsolásának hatását a 4.3.5. fejezetben elemzem. Bekapcsolt titkosítás mellett rejtéskor egy CipherInputStreamen futtatom át az elrejtendő fájlt, míg visszahozáskor egy CipherOutputStreamet használok, így a titkosítás mint pluszszolgáltatás meglehetősen csekély változást visz be a kódba.

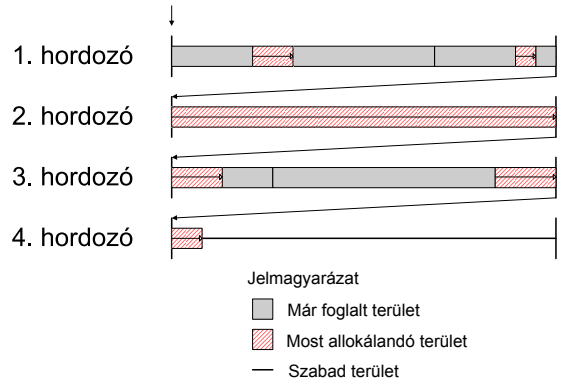
A rejtjelezés után az adatokat a tárolásért felelős rész kapja meg. Ez egy rétegekre osztott struktúra, melyben minden réteg más-más szinten absztrahálja a hordozókban található szteganografikus kapacitást. A 3.1. ábrán látható a rétegek hierarchiája. A legalacsonyabb szint, a *fizikai réteg* felel a hordozó adatrejtési helyeinek eléréséért. Egy raszteres képben például adatrejtési pozíciónak tekinthetünk egy színinformációt leíró bájtot, melynek LSB-jét adatrejtésre szeretnénk felhasználni. JPEG képnél például adatrejtési hely lehet a DCT mátrix egy eleme. A fájlok elérését a Java NIO architektúra [27] memóriához rendelt puffereinek segítségével valósítottam meg.

Eggyel felsőbb szint az *algoritmusréteg*, mely a hordozó által biztosított információrejtési pozíciók fölé egy algoritmust illeszt. Például egy raszteres képbe többféleképpen is elrejthetünk adatot. Implementálható algoritmusrétegbeli objektum a sima LSB adatrejtés, ahogy az általam javasolt normaértékes adatrejtés is. Ez a réteg egy folytonos tárterületnek láttatja a felsőbb rétegeknek az általa kezelt hordozó szteganografikus kapacitását, a megvalósított algoritmuson keresztül. Magától értetődő, hogy ez a szteganografikus kapacitást alkotó helyek egyfajta sorrendezését is biztosítja az algoritmusréteg (így például érdemes lehet a biztonság növelése érdekében álvéletlen sorozat mentén meghatározni az adatrejtési helyeket [20], mely sorozat magját mint sztegokulcsot a felhasználó adja meg a grafikus felületen beírt jelszóval). Más szavakkal, az algoritmusrétegbeli objektum kölcsönösen egyértelmű hozzárendelést biztosít a hordozó szteganografikus kapacitásához.

A legfelső szint az *I/O réteg*, mely az algoritmusrétegbeli objektumok által leképezhető kapacitásokat összefogja egy egységes tárterületté. Ez a réteg szolgál interfészként a külvilág számára. A stratégiát, amely szerint a hordozók által biztosított kapacitáson el kell osztani az elrejtett adatokat, egy mellérendelt réteg, az *allokátor* határozza meg. Ilyen stratégia lehet pél-

dául a lineáris felhasználás, amikor egészen addig írunk egy hordozóba, míg abban van szabad szteganografikus kapacitás. Egy másik módszer a „terheléskiegyenlítés”, amikor igyekszünk minden hordozóba nagyjából egyforma mennyiségű adatot tenni. Ez a biztonságra pozitív hatással lehet (hiszen minden hordozón közel azonos mérvű transzformáció zajlik), azonban a metainformációk mennyiségét drasztikusan megnövelheti a fájldarabok megszaporodása miatt. Bármi legyen is az allokációs stratégia, az allokátor feladata nyilvántartani a foglalt kapacitásokat és megadni a stratégia szerint következő rejtési helyet az I/O réteg számára. Fontos megjegyezni, hogy egy már meglévő stegodrive-on az allokációs stratégia megváltoztatása nem jár semmilyen kockázattal, tekintve, hogy az allokátor csak az újonnan bekerülő adatokra van befolyással. A régebben lefoglalt területek ugyanis már darab rekordok formájában el vannak tárolva az adatbázisban.

A programban egyelőre kizárólag lineáris allokáció van implementálva (lásd a 3.6. ábrát). Ennek lényege, hogy a hordozókat sorrendbe állítjuk, és végigmenve rajtuk mohó módszerrel lefoglaljuk a talált szabad területeket. Az algoritmus addig fut, míg az allokált kapacitások összege el nem éri a kért értéket. Az allokáció legkisebb egysége a jelenlegi implementációban bájt.



3.6. ábra. A lineáris allokáció bemutatása egy 4 hordozót tartalmazó stegodrive-on

3.5. A rétegek implementációja

A fizikai rétegbeli objektumok a PhysicalLayer interfészt valósítják meg. Ennek fontosabb metódusait foglalja össze a 3.1. táblázat. Az abszolút címzéses metódusokra általában nincs szükség, mivel használatukkor a rétegek közti felelősségmegosztás az adatrejtési algoritmus megvalósításában sérül. Az adatrejtési egységek írásakor, illetve olvasásakor a hely a hordozó típusától függ, míg az egyéb függvények számára megadott ofszet bitben értendő. Ennek az az oka, hogy sok szteganográfiai algoritmus számára az elrejtett adat egysége a bit. Mindazonáltal, tekintve, hogy más felhasználásokhoz viszont a bájtszervezés a praktikus, implementálásra kerültek egész bájtot elrejtő függvények is, melyek funkciója pusztán a bitenkénti rejtésre való visszavezetés. A megadott függvények mindegyike dobhat StegoFileAccessExceptiont, például túlcímzés vagy értelmezhetetlen adatrejtési egység esetén. Mindazonáltal az algoritmusréteg helyes működése esetén túlcímzés nem léphet fel, vagyis a kivétel pusztán a biztonságosabb hibafelderítést szolgálja.

Az algoritmusrétegbeli objektumok a 3.2. táblázatban foglalt metódusokat valósítják meg. Az ofszetek itt már mindig bájtokban értendőek, mivel az általános I/O műveletek legkisebb egysége tipikusan a bájt. A kapacitás is bájtokban értendő, ebből adódóan egy hordozóban legfeljebb 7 bitnyi szteganografikus kapacitás kihasználatlanul maradhat.

Az I/O rétegbeli objektumok a 3.3. táblázatban foglalt interfészen[6] keresztül látszanak. Ez a réteg már teljes rejtett vagy elrejtendő fájlok szintjén működik, ezért nincs olyan tagfüggvény, melynek ofszetet kellene megadni. A rejtett fájlok darabjainak hozzáadására és törlésére szolgáló függvények azért szükségesek, mert az allokátornak is tudnia kell a stegodrive-on levő darabokról. Magára az allokátorra is lehetett volna bízni a darabok kiolvasását az adatbázisból, mindazonáltal az objektumorientáltság felelősségmegosztási filozófiájából adódóan ezt a megvalósítást tartottam helyesnek és pontosnak.

Az allokátor által megvalósított interfészt írja le a 3.4. táblázat. A legfontosabb függvény a *getNextPositions*, ennek segítségével kérhetünk ugyanis az allokátortól információt, hogy

[6] Az I/O réteg interfészét azért definiáltam, mert létezhetnek olyan műveletek (pl. tömörítés, rejtjelezés), melyek csak fájlszinten végezhetőek el hatékonyan. Ebből adódóan hoztam azt a tervezői döntést, hogy a különböző képességekkel felruházott I/O rétegbeli osztályok egy közös interfészen keresztül legyenek láthatóak.

Metódus	*Funkció*
byte getAbsOffset(int offset)	Bájt kiolvasása a hordozóból abszolút címzéssel.
void getAbsOffset(int offset, byte[] info)	Bájtok kiolvasása abszolút címzéssel, a paraméterként kapott tömb hosszán.
byte getStegoOffset(int offset)	Egy bájtnyi rejtett adat kiolvasása a megadott ofszetről.
void getStegoOffset(int offset, byte[] info)	Rejtett bájtok kiolvasása a megadott ofszettől kezdődően, a kapott tömb hosszán.
boolean getStegoBit(int offset)	Egyetlen rejtett bit kiolvasása a megadott ofszetről.
void setStegoBit(int offset, boolean info)	Egyetlen bit elrejtése a megadott ofszetre.
void setAbsOffset(int offset, byte data)	A kapott bájt hordozóba írása a megadott ofszettől kezdődően, abszolút címzéssel.
void setStegoOffset(int offset, byte data)	A kapott bájt elrejtése a megadott ofszettől kezdődően.
Object getHidingUnit(int offset)	Adatrejtési egység kiolvasása a megadott pozícióról. A metódus által visszaadott adat hordozótípusonként változó értelmű. Raszteres kép esetén ez egy pixel egy színkomponensét jelenti.
void setHidingUnit(int offset, Object setHidingUnit)	Adatrejtési egység írása a megadott pozícióra. A metódus paramétere hordozótípusonként más típusú lehet. Raszteres kép esetén ez egy pixel egy színkomponensét jelenti.

3.1. táblázat. A fizikai réteg által megvalósított tagfüggvények

Metódus	*Funkció*
int getCapacity()	Hordozó kapacitásának kiszámítása.
void write(int offset, byte info)	A megadott bájt elrejtése a megadott ofszetre.
void write(int offset, byte[] info)	A megadott bájtok elrejtése a megadott ofszettől kezdődően.
byte getByte(int offset)	Egy rejtett bájt kiolvasása a megadott ofszetről.
void getByte(int offset, byte[] info)	A kapott tömb feltöltése rejtett bájtokkal a megadott ofszettől kezdődően.
void setKey(byte[] key)	Sztegokulcs beállítása.
void setSlider(int value)	Biztonság-kapacitás kompromisszum megadása.

3.2. táblázat. Az algoritmusréteg által megvalósított tagfüggvények

Metódus	*Funkció*
void setKey(byte[] key)	Titkosítási, illetve sztegokulcs beállítása. Az I/O réteg feladata a kulcsot szétterjeszteni az algoritmusrétegbeli objektumpéldányok közt.
void setSlider(int value)	Biztonság-kapacitás kompromisszum beállítása. Az I/O réteg feladata a megadott értéket szétterjeszteni az algoritmusrétegbeli objektumpéldányok közt.
void newAlgorithm(Cover cov)	Új algoritmusrétegbeli objektumpéldány létrehozása a megadott hordozóhoz
void setAllocationStrategy(String allocationStrategy)	Allokációs stratégia megadása. Jelenleg egyedül a „linear” van implementálva.
void getFile(String filename, Piece[] pcs, String whereto)	Adott nevű rejtett fájl kimentése a megadott darabokból a megadott helyre
void putFile(String filename)	A megadott fájl elrejtése
void deleteFile(String filename)	A megadott rejtett fájl törlése
void addPiece(Piece pc)	Rejtettfájldarab hozzáadása.
void removePiece(Piece pc)	Rejtettfájldarab törlése
int free(Cover cov)	Hordozó szabad szteganografikus kapacitása az érvényben levő norma mellett
int capacity(Cover cov)	Hordozó teljes szteganografikus kapacitása az érvényben levő norma mellett

3.3. táblázat. Az I/O rétegben megvalósított tagfüggvények

hogyan „képzeli el” adott számú bájt lefoglalását. A függvény használatakor a visszaadott darabokat nem jegyzi meg az allokátor, tekintve, hogy a mögöttük levő tartalom még nem került elrejtésre – az I/O rétegre van bízva a valóban kiírt tartalomhoz köthető darabok elhelyezése az allokátorban.

Metódus	*Funkció*
void addCover(Cover cov, int cap)	Hordozó definiálása az allokátor számára.
void addPiece(Piece pc)	Rejtettfájldarab hozzáadása
void removePiece(Piece pc)	Rejtettfájldarab felszabadítása
Piece[] getNextPositions(int length)	Adott hosszt lefedő darabhalmaz kiadása
int free(Cover cov)	A hordozóban levő szabad hely kiszámítása.

3.4. táblázat. Az allokátor tagfüggvényei

4. fejezet

Analízis

A következőkben a stegodrive-ot mint koncepciót, illetve a benne alkalmazott algoritmusokat fogom vizsgálni.

4.1. A javasolt normaértékes LSB algoritmus elemzése

A javasolt algoritmus tehát 3-szor 3-as blokkokra osztja a képet, és minden blokk középső pixelébe elrejt színkomponensenként 1 bitnyi adatot, ha a blokk szórása az adott színkomponensre nézve egy megadott küszöb felett van. Ennek komplexitás, kapacitás és biztonság szempontjából egyaránt vannak implikációi, melyeket az egyszerű LSB algoritmussal fogok szembeállítani.

4.1.1. Kapacitás

Az algoritmus értelemszerűen erős kapacitásdegradációt okoz a sima LSB adatrejtéshez képest. Ez kétféle forrásból származik:

- Egy 9 képpontos blokkból egyetlen pixelt használunk fel.
- Nem feltétlenül oszthatóak a kép dimenziói 3-mal, és a maradék elvész.

Az első kapacitásveszteség egész blokkokra vonatkozik, így közelítőleg $8 : 9$ arányú az egész képre vetítve. Pontosan kifejezve ez a veszteség

$$\frac{8}{9} \cdot \left\lfloor \frac{H}{3} \right\rfloor \cdot \left\lfloor \frac{W}{3} \right\rfloor \tag{4.1}$$

bit színsíkonként. A szorzat első tagja azt reprezentálja, hogy egy teljes 3-szor 3-as blokkba egyetlen bitet rejtünk 9 helyett, míg a szorzat második és harmadik tagja a magasságban és szélességben kialakítható pixelhármasok számát jelenti.

A degradáció második forrása által okozott veszteség pixelre pontosan

$$\left\lfloor \frac{H}{3} \right\rfloor \cdot (W mod\, 3) + \left\lfloor \frac{W}{3} \right\rfloor \cdot (H mod\, 3) + (W mod\, 3) \cdot (H mod\, 3) \tag{4.2}$$

bit színsíkonként, ahol W a kép szélessége, H a magassága. Az összeg első tagja a széltében, második tagja a hosszában fel nem használható pixeleket reprezentálja, míg az utolsó tag a kép „sarkában” maradt pixelek száma.

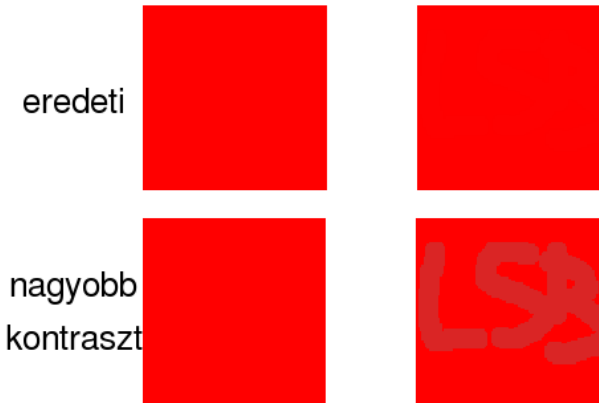
Az egész képre vetített veszteség a 4.1. és a 4.2. összefüggések összege 3-mal szorozva, ha 24 bites BMP képről beszélünk. Belátható továbbá, hogy ha a kép négyzet alakú n oldalhosszal, akkor az első típusú veszteség nagyságrendileg n^2-re, míg a második típusú veszteség n-re adódik, ha n ∞-hez tart.

4.1.2. Biztonság

Az algoritmus elsősorban az emberi érzékeléssel szembeni biztonságot tűzi ki célul. Demonstrációként tekintsük a 4.1. ábrát.

Az ábra felső sorában baloldalt egy homogén piros négyzet látható, míg jobboldalt egy olyan piros négyzet, melyben egyes pixelek LSB-it megváltoztattam[1]. A különbség TFT-LCD monitoron egyáltalán nem észrevehető, ellenben katódsugárcsöves megjelenítőeszközön megfelelő színbeállítások mellett már észlelhető az eltérés. (Ha minden pixel minden szín-

[1]Nem valódi üzenetet rejtettem el a négyzetben, hanem a GIMP képszerkesztő programmal belerajzoltam egy mintát egy, a teljesen pirostól csak 1-1 bitben eltérő színnel. Az ábra célja pusztán a hatásmechanizmus illusztrálása.



4.1. ábra. LSB-változtatás észlelhetősége

komponensének 2 LSB-jét használjuk fel, a különbség már TFT-LCD monitoron is látható helyes beállítások mellett.) Ennek magyarázata, hogy az emberi szem egy homogén felületen könnyebben észreveszi a változást mint egy heterogénen. Bármilyen monitoron eklatánsan jelentkezik azonban a különbség, ha a kép kontrasztját drasztikusan megnöveljük egy képszerkesztő program segítségével (a példához a GIMP nyílt forráskódú szoftvert használtam).

Ugyanezen kontrasztmanipulálási módszerrel megnéztem továbbá, hogy az adaptivitási normának milyen hatása van a végeredményre. Az eredeti fénykép a 4.2. ábrán látható. A kontrasztot drasztikusan megnövelve a kép jobb felső sarka a 4.3. ábra szerint alakul. Ha a javasolt algoritmust 0 szórási küszöbbel futtatjuk úgy, hogy a kép szteganografikus kapacitását teljesen kihasználjuk, akkor viszont a 4.4. ábrán látható képet kapjuk. A különbség egyáltalán nem nevezhető szembetűnőnek, mindazonáltal árulkodóak a szorosan egymás alatt elhelyezkedő liláskék pixelek, például a piros színnel megjelölt helyeken. A 4.5. ábra már a 15-ös küszöb hatását mutatja meg. Itt már jóval nehezebb különbségeket észrevenni az eredeti és a rejtett adatot tartalmazó kép közt. Továbbá szemmel majdnem teljesen esélytelen észrevenni az előbbiekhez hasonló módon a szteganográfia jelenlétét. Mindezért viszonylag nagy árat fizetünk: a kép kapacitása 42481 bájtról 11300 bájtra esett vissza.

A szteganográfia vizuális észlelhetősége mellett szintén nagyon fontos tényező a statisztikai alapú szteganalízissel szembeni biztonság. Ezen a téren egy általános jellegű korlátba ütközünk: az LSB-jellegű szteganográfiai algoritmusok szteganalízise ma már meglehetősen eredményes. [20] alapján alapvetően három tényezőtől függ a szteganográfia detektálhatósága:

4.2. ábra. Az eredeti fénykép

4.3. ábra. Az eredeti fénykép jobb felső sarka, nagy kontraszttal

4.4. ábra. A rejtett adattal teleírt kép jobb felső sarka (0-s küszöb mellett)

4.5. ábra. A rejtett adattal teleírt kép jobb felső sarka (15-ös küszöb mellett)

- Elrejtett üzenet hossza a hordozó hosszához képest
- Elrejtett üzenet megoszlása (szekvenciális, álvéletlen stb.)
- A hordozó színeinek száma

Az első pont tekintetében az algoritmus bizonyos előnnyel indul a sima LSB-hez képest, tekintve, hogy a kép méreteivel négyzetesen arányos veszteséget visz be az eljárásba (lásd a 4.1.1. szakaszt). A második pontot tekintve se pró, se kontra nem állíthatunk semmit, mivel mindkét algoritmust képesek vagyunk álvéletlen sorozat mentén futtatni a biztonság javítása érdekében, ahogy ez meg is történik a normaértékes módszer implementációjában. A harmadik pontra nézve szintén nincs kihatása az algoritmusnak – a felhasználó annyival járulhat hozzá a biztonság növeléséhez, hogy nagy színhűségű szkennerrel vagy fényképezőgéppel készült fényképet használ hordozónak, és kerüli például a számítógéppel készített ábrákat.

4.1.3. Komplexitás

A javasolt algoritmus az egyszerű LSB rejtéshez képest megnövekedett időkomplexitással bír. Ennek az az oka, hogy az egyszerű aritmetikai műveleteknél bonyolultabb számításokat kell alkalmazni a szórás kiszámításához (pl. gyökvonás). Ha meg akarjuk keresni a raszteres képben az n-edik ofszethez tartozó rejtési pozíciót, akkor ezeket a számításokat legalább n-szer el kell végezni – ez a szám azonban drasztikusan megnőhet, ha az adaptivitási norma nagy szórást ír elő, mert megugrik a rossz blokkok száma.

Az első, „naiv” implementáció, melyben minden egyes ofszethez külön-külön számítottam ki a rejtési pozíciót, éppen ezért szélsőségesen lassú lett: kevesebb mint fél kilobájt adat elrejtése egy fényképbe majdnem 8 percet vett igénybe. A végleges implementáció ezért az elsőnek lekérdezett rejtési pozíció kiszámításakor egy keresési táblát hoz létre egy HashMap objektum formájában. A későbbi lekérdezésekkor így nem szükséges újra elvégezni a költséges számításokat, és a már említett adat elrejtése kevesebb mint 1 másodperc alatt lezajlik.

4.2. A stegodrive mint koncepció elemzése

A stegodrive-ot többféle szempontból is célszerű elemezni. Ebben az alfejezetben a különféle képességű támadókkal szembeni biztonságot és a teljesítményt fogom vizsgálni.

4.2.1. Teljesítmény

A stegodrive a jelenlegi állapotában pazarlóan bánik a memóriával. Az implementáció ugyanis igencsak naivan hasznosítja a memóriába leképzett puffereket, melyek – mint az tapasztalati úton kiderült – meglehetősen drága erőforrásokat képeznek. A jelenlegi megvalósításban az összes hordozó egyszerre kerül megnyitásra; ez még kis kapacitású meghajtó esetén is könnyen fel tudja emészteni az operatív tárat[2].

Megoldást jelenthet a problémára, ha a fizikai rétegbeli objektumok interfészére felveszünk egy *open()–close()* párost, és ezeket az I/O rétegbeli objektum működteti az allokátor által meghatározott stratégia szerint. Ez némiképp megemeli a szoftver processzorigényét és I/O-műveleteinek számát, viszont az áldozat bőségesen megtérül a nyert memórián. Elviekben ugyanis az ilyen módon megvalósított stegodrive maximális memóriaigénye arányos a legnagyobb felhasznált hordozó méretével, tekintve, hogy a lefoglalt tárterületet explicite fel lehet szabadítani a puffer lezárásával, és nem kell megvárni, míg a Java szemétgyűjtő algoritmusa teszi meg ezt helyettünk.

[2]Ez annak ellenére így van, hogy a memóriába leképzett puffer voltaképp analóg az operációs rendszer által használt virtuális memóriával, vagyis a puffer által lefoglalt memóriaterület „mögött” levő tartalom általában nincs egyszerre betöltve a háttértárról.

4.2.2. Támadó modell

A bevezető fejezetben láttuk, hogy egy szteganográfiára alapuló rendszerrel támasztott leggyakoribb követelmény az információrejtés észlelhetetlensége – és nincs ez másképp a stegodrive esetében sem. Az elemzést az alábbi képességekkel rendelkező támadókra vagy ellenfelekre végzem el:

Egyszeri támadás : a támadó egyetlenegy időpillanatban képes megfigyelni a hordozók által használt területet.

Csak hordozó : a támadó képes többször megfigyelni a hordozóként használt fájlokat tartalmazó tárterületet.

Hordozó és metainformáció : a támadó képes többször megfigyelni a hordozók és a metainformációs adatbázis tárolására használt tárterületet.

Egész rendszer : a támadó képes többször megfigyelni a stegodrive-ot futtató egész rendszert, beleértve az elemi I/O műveleteket, a memóriatartalmat stb.

Látható, hogy ezek a támadási formák egyre erősebb ellenfeleket tételeznek fel. Fel kell viszont tételeznünk, hogy a támadó sosem képes olyan alkalmakkor „lefényképezni” a rendszert, ha épp fut rajta a stegodrive, és van beolvasott adatbázis. Ez ugyanis azt jelentené, hogy az ellenfél tudomást szerezhet arról, hogy milyen adatbázison fut a stegodrive, és hogy milyen sztegokulcsot adott meg a felhasználó. Nyilvánvaló, hogy az ilyen képességű támadók ellen a stegodrive nem nyújt védelmet, de ez nem a szoftver gyengeségéből, hanem pusztán azon egyszerű tényből ered, hogy bármilyen kriptografikus vagy szteganografikus fájlrendszer csak lecsatlakoztatott állapotában biztonságos – hasonlóan például egy páncélszekrényhez, mely semmiképp sem védi meg a beletett értékeket, ha az ajtó nyitva van, mert a széf legitim felhasználója éppen kivesz vagy betesz oda valamit.

A felsorolt támadók közül az egész rendszert többször megfigyelni képes ellenfél tudhat meg a legtöbbet. Az elemi I/O műveletek szekvenciájának naplózása révén például képes lehet megfigyelni, hogy ha összehasonlítja az azonos abszolút ofszetekről kiolvasott majd oda

visszaírt értékeket, a változás mindig legfeljebb a legjelentéktelenebb bitet érinti. Ráadásul hiába végezzük álvéletlen sorozat mentén a rejtést, hiszen a támadó kezében van az az ofszet-szekvencia, amelyen a módosítások történtek – ilyen formán nemcsak, hogy képes értesülni a szteganográfia jelenlétéről, de kikapcsolt titkosítás esetén az elrejtett fájlt egyszerűen kiolvashatja. Az I/O műveleteket folyamatosan naplózó ellenféllel szemben tehát majdhogynem lehetetlen védekezni. Ha az I/O rétegben véletlenszerű szekvencia mentén olvassuk be az elrejtendő adatot, akkor legalább az üzenet tartalmát megóvhatjuk. Erre a célra további védekezési lehetőség [25] az álolvasások-álírások (nem valós rejtési igényhez tartozó, hanem direkt a támadó megzavarására alkalmazott, funkcionálisan jelentéktelen I/O műveletek) beiktatása.

A hordozó-, illetve metainformációterületet többször megfigyelni képes támadó esete már valamivel egyszerűbb. Az ilyen ellenfél képes többször „lefényképezni" az említett tárterület egész tartalmát, és összehasonlításokat végezni az így nyert adathalmazok közt. A támadó a gyakori stegodrive-használat egyértelmű jeleként egy idő után rá fog akadni a gyakran módosított fájlok listájában a metainformációs adatbázisra, amely – tekintve, hogy tartalmazza minden rejtett fájl nevét és a felhasznált hordozókat – azonnal elárulja a szteganográfia jelenlétét. Ez alkotja a stegodrive koncepció jelenlegi legnagyobb hiányosságát: a metainformációs adatbázist nem védi semmi. A felhasználónak ezért csak akkor van esélye egy ilyen képességű ellenféllel szemben, ha használ valamilyen másik szteganografikus fájlrendszert, például egy nagy replikációs tényezővel inicializált StegFS-t[3].

A pusztán a hordozókat tartalmazó területet megfigyelni képes támadó ellen már viszonylag nagy eséllyel veszi fel a harcot a stegodrive. Az ellenfélnek ugyanis ilyenkor pusztán a változó hordozókról állnak rendelkezésre adatai[4]. Elég könnyen adódik, hogy melyiküket használja a felhasználó gyakrabban, ha a fájlrendszer rögzíti a módosítás időpontját. Ezért a biztonság szavatolása érdekében mindenképp úgy kell felcsatolni az állományrendszereket, hogy a módosítási időpontok ne kerüljenek feljegyzésre (pl. noatime opció Linux alatt). Azon-

[3]Ez nem teszi értelmetlenné a stegodrive használatát, ugyanis, mint azt már láttuk, a StegFS egy veszteséges fájlrendszer, míg a stegodrive nem az. Mindazonáltal az adatbázis szteganografikus tárolását mindenképp meg kell oldani a későbbi implementációkban.

[4]Nem nehéz elképzelni egy ilyen élethelyzetet, ha a felhasználó egy olyan pendrive-on tárolja az adatbázist, melyről a támadó nem képes tudomást szerezni.

ban még ilyenkor is az ellenfél rendelkezésre állnak a „fényképek” a tárterület állapotáról, és egy alapos támadóban felmerülhet a gyanú, ha például azt látja, hogy egy kép tartalma módosult, de szemmel nem látszik rajta változás. Az ilyen képességű támadások ellen jobban teljesít a StegFS a stegodrive-nál: mivel előbbi egy gazdafájlrendszer szabad területére rejti el az információt, természetesebbnek hat, ha ott „szemét” gyűlik fel a „rendeltetésszerű használat” során, míg az nehezebben megmagyarázható, hogy miért mennek végbe emberi szemmel nézve jelentéktelen módosítások egy fájlon.

Az utolsó, egyben leggyengébb ellenfél csak egyetlen állapotában képes a rendszert megfigyelni. Ilyen jellegű támadás például a hordozókat tartalmazó meghajtó eltulajdonítása vagy hatóság általi legfoglalása. Intuitívnak hat a feltételezés, miszerint az ilyen támadási forma mind közül a leggyakoribb: míg az előzőekhez például kémprogramok telepítése szükséges a felhasználó számítógépére, addig ehhez a módszerhez nincsen szükség olyan sok „aknamunkára”. Mindazonáltal ilyenkor a legnehezebb a támadó dolga: csak a lefoglalt meghajtón aktuálisan megtalálható adatok állnak a rendelkezésére. Egyértelmű tehát, hogy – mivel nem építhet támadási stratégiát a tartalom gyanús változásainak megfigyelésére – csak vizuális vagy statisztikai szteganalízissel kezdhet neki a szteganográfia felderítésének.

4.2.3. Az allokációs stratégia hatása a biztonságra

Az allokátor tulajdonságairól eddig csak pár szó esett. Intuitív feltevés azonban, hogy az adatok elhelyezkedése a hordozócsoporton belül akár jelentős hatással is lehet a szteganografikus biztonságra. A *kötegelt szteganalízissel* foglalkozó [26] igazolja ezt a feltevést: azt állítja, hogy azonos jellegű forrásból származó (másképpen fogalmazva: azonos eloszlású) hordozók esetén a rejtés akkor a legbiztonságosabb, ha egyenletesen van elosztva a rejtett információ a hordozócsoportban. A legkönnyebben felismerhető stratégia pont a lineáris allokáció, mely a jelenleg egyetlen implementált módszer. Mindazonáltal a stegodrive-koncepció lehetővé teszi akármilyen allokációs stratégia alkalmazását, vagyis a biztonságosabb „terheléskiegyenlítéses” algoritmus komolyabb nehézségek és strukturális átalakítások nélkül implementálható.

4.3. Az adatrejtés hatása a statisztikai tulajdonságokra

Bár a javasolt algoritmus eredeti célja a „vizuális szteganalízissel” szembeni biztonság, érdekesnek láttam megvizsgálni a statisztikai tulajdonságokat is. Módszertanként [20] szolgált. Az itt leírt módszer a közeli színpárok számát használja mérceként. Az előfeltevés ugyanis az, hogy a hordozóként szóbajövő képek nem használják ki az egész színteret, különösen, ha például tömörített képekből lettek visszatranszformálva tömörítetlen formátumba. Mindez azzal jár, hogy az LSB-típusú algoritmusok megnövelik a szomszédos színpárok számát. A módszertan szomszédos színpárnak definiálja (R_1, G_1, B_1) és (R_2, G_2, B_2) színeket (R_i, G_i, B_i 0-tól 255-ig vehetik fel értékeiket az egész számok halmazán, inkluzíve), ha teljesül a 4.3. összefüggés.

$$(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2 \leq 3 \tag{4.3}$$

Szemléletesen megfogalmazva, ez a képlet egy „szomszédossági kockát” definiál az RGB színtér által kifeszített kockában. Ha a vizsgált szín egyik eleme sem 0 vagy 255, akkor egy, a szín komponensei által kijelölt középpontú $3 \times 3 \times 3$-as kocka lesz azon koordináták helye, melyek kielégítik a 4.3. összefüggést. Ha bármelyik színkomponens az intervallum szélén levő értéket veszi fel, akkor a szomszédossági kocka alakja megváltozik. A legelfajultabb esetben, amikor az RGB kocka sarkában van a vizsgált szín, egy $2 \times 2 \times 2$-es kockára szűkül a szomszédossági helyek halmaza.

A szteganalízis menete a következő:

1. Összes színpár és szomszédos színpárok megszámolása a hordozóban.

2. $\alpha \cdot 3 \cdot H \cdot W$ bit méretű tesztüzenet elrejtése. Itt H a kép magassága, W a szélessége, α pedig körülbelül 5%. A felhasznált cikk szerint ez az érték valamelyest javíthat a szteganalízis hatékonyságán, ha az elrejtett üzenet hosszának megfelelően szabályozzuk, de a változás nem számottevő. Ezért a módszert konstans α mellett alkalmaztam.

3. A módosult hordozóban az összes színpár és a szomszédos színpárok megszámolása.

4. $R = \frac{\frac{U'}{A'}}{\frac{U}{A}}$ számítása, ahol U az összes színpár, A a szomszédos színpárok száma, míg ezek vesszős változatai ugyanezen mennyiségek a tesztüzenet beágyazása után. Arra számítunk, hogy R üres hordozókra nagy lesz, míg már eredetileg is rejtett üzenetet tartalmazóakra kicsi.

5. Eredmény összevetése a 4.1. táblázattal. A táblázat utolsó sora például azt jelenti, hogy a hordozó kapacitása vélelmezhetően teljes egészében ki van használva, ha $R \leq 1{,}0028$

Felhasznált hely	*Küszöb*
1%	$1{,}1606$
5%	$1{,}0935$
10%	$1{,}0506$
20%	$1{,}0206$
50%	$1{,}0059$
100%	$1{,}0028$

4.1. táblázat. A felhasznált szteganalízis módszertan küszöbértékei

A módszer természetesen hajlamos hibázni (hibás pozitívokat és hibás negatívokat produkálni), különösen, ha alacsony kihasznált kapacitást feltételezünk. A cikkben ismertetett mérések szerint 1%-os kihasználtságnál a hibaarány 40%, míg 100%-hoz közel pusztán $1{,}1\%$.

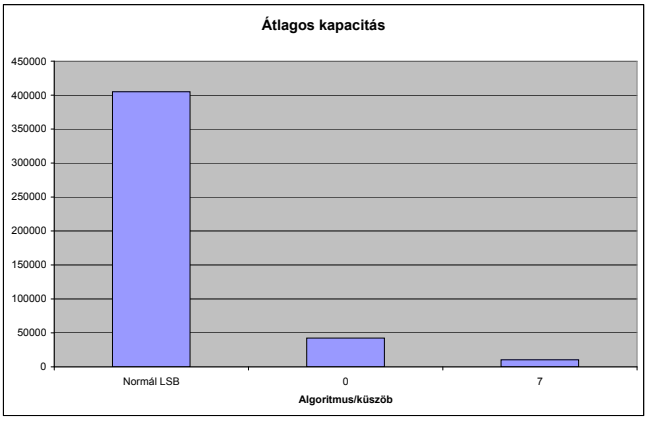
4.3.1. A módszertan kiterjesztése

A javasolt normaértékes adatrejtési sémának van egy érdekes tulajdonsága: megjósolható helyekre helyezi el az elrejtett adatot. Ezért a módszertant kiterjesztettem: vizsgáltam a szomszédos színek számát pusztán az algoritmus által potenciálisan használható pixelek halmazán is. A tesztüzenetet – melynek hosszát az adott szórásküszöb melletti átlagos kapacitás 5%-ára méreteztem – mindig a tesztelt küszöbérték mellett ágyaztam be, lineáris allokációval. Mindezzel arra kerestem a választ, hogy a módosított szteganalízis hatékonyabb lesz-e az így elrejtett adatokra.

4.3.2. A felhasznált minta

Mintaként 75 darab, 1200×900 pixeles, 24 bites BMP kép szolgált, melyek JPEG formátumról lettek átkonvertálva. A rejtésnél az adott normával definiált szteganografikus kapacitást teljesen kitöltöttem bináris, ámde entrópia szempontjából véletlenszerűnek nem mondható állományokkal (az entrópiát természetesen befolyásolja, hogy a rejtés előtti titkosítás be van-e kapcsolva).

Az átlagos kapacitás a 4.6. ábrán látottaknak megfelelően alakult.



4.6. ábra. Átlagos kapacitás (bájtban)

Látható, hogy a 0-s szórásküszöbhöz nagyjából a 4.1.1. fejezetben írtakhoz mérhető kapacitásveszteség tartozik, utána pedig – a homogén blokkok kizárása miatt – további, ámde jóval kisebb mérvű csökkenés jelentkezik.

4.3.3. A statisztikák elkészítésének módja

Míg a kapacitásstatisztikát egyszerűen a stegodrive *Szabad terület* gombjának használatával kérdeztem le, a szomszédos színpárok megszámolásához már segédprogramokra volt szükségem.

Az első a tesztüzenet elrejtéséhez szükséges: funkciója egyszerűen annyi, hogy vagy sima

LSB adatrejtéssel, vagy a javasolt algoritmussal lineáris allokációs stratégiát követve elrejti azt a sorozatot, melyet az ASCII tábla lehetséges értékeinek a kívánt üzenetméretnek megfelelő számú egymás után írásával kapunk[5]. A programnak megadható a felhasználandó algoritmus, a hordozó elérési útja, az elrejtendő bájtok száma, valamint a normaértékes rejtési algoritmus esetén a szórásküszöb. Ezeket a paramétereket a parancssoron kell megadni, indításkor.

A második szoftver a hordozó statisztikai vizsgálatát segíti. Funkciói közé tartozik a színek és a szomszédos színpárok megszámolása, valamint a gyakorisági grafikon származtatásához szükséges lista elkészítése. Mindezeket képes az egész képre vagy csak a normaértékes algoritmus által használt pixeleken kiszámítani. Ezen kívül lehetőség van a színinformációkat egyszerűen 0-tól 255-ig terjedő számok formájában kiíratni, szintén lehetővé téve a csak a blokkcentrumban levő pixelek figyelembe vételét. Ez a funkció – bár végül nem használtam fel – bizonyos esetekben könnyebbé teheti a statisztikai elemzést, például egy közelítő eloszlásfüggvény származtatásakor[6]. A program utolsó funkciója a több kép gyakorisági adatait tartalmazó fájl „összenyomása”, vagyis egy olyan fájl származtatása, mely az összes hordozóra vonatkozó gyakorisági információval szolgál.

Mindkét programot kötegelt módon futtattam a linuxos világban általánosan meglévő *find* parancs használatával, a következő szintaxissal[7]:

```
find . -name '*.bmp' -exec java -jar BMPStatistics.jar
funkcio {} eredmeny.txt \;
```

A „funkcio” szó helyén a használni kívánt műveletet kell megadni. Ez lehet *uniq*, *freq*, *dump*, valamint ezek *all* utótaggal kiegészített változatai, attól függően, hogy csak a blokkcentrumban elhelyezkedő pixelekre, vagy az összes pixelre szeretnénk elvégezni az egyedi színek és a szomszédos színpárok megszámolását, a gyakorisági adatok lekérdezését, illetve

[5] A módszertant ismertető cikk sem a tesztüzenet milyenségéről, sem annak elhelyezéséről nem ír. Ezek megválasztása saját tervezői döntésem eredménye volt.

[6] Mindazonáltal a funkció használata számottevő háttértárkapacitást követel meg nagyobb méretű hordozó esetében, és viszonylag sokáig tart.

[7] Itt a {} páros a *find* parancsnak megadott szabályra illeszkedő állomány nevét jelenti, míg \; azt jelzi a *find* számára, hogy itt véget érnek a futtatandó programnak megadott paraméterek. Az eredményt tartalmazó szövegfájl neve azért nem változik, mert úgy láttam célszerűnek a program implementációját, hogy már létező eredményfájl esetén nem felülírás, hanem hozzáírás történik. Mindez jelentősen megkönnyítheti az olyan statisztikák elkészítését, melyek az összes hordozóról adnak információkat.

a színinformációk számértékekkel történő reprezentálását. A gyakorisági összegzést a *sum* funkció biztosítja, ennek értelemszerűen nincsen *all*-os párja.

4.3.4. Szteganalízis titkosítatlan rejtett információra

A szteganalízis elvégzésének első lépése a módszertan letesztelése volt. Kíváncsi voltam, hogy az üres hordozókra hogyan alakul a raszteres képekben a „találati arány". Az eredmény 2 hamis pozitív a 75-ből. Ebből arra a következtetésre jutottam, hogy a módszer megfelelően működik, amit az a tény is erősített, hogy a bizonytalanság ezekben az esetekben is elérte a 40%-ot.

A következő lépés a egyszerű LSB algoritmusnak, mint a felhasznált szteganalízis módszertan referenciájának tesztelése volt. Ehhez minden hordozót teleírtam a segédprogrammal (miután a kép dimenziójából előzetesen kiszámítottam a kapacitást), majd megvizsgáltam a szomszédosszín-statisztikát. Az eredmény 100%-os helyes pozitív arány. A hordozókon mért R érték átlaga $1{,}000183$ lett, míg maximuma $1{,}002351$. Összevetésképpen: ugyanezek a számok üres hordozók esetén $1{,}334155$ és $1{,}617955$ lettek, és míg üres esetben a minimum $1{,}054294$ volt, a teleírt hordozóknál 31 darabnál mértem 1-nél kisebb értéket. Tekintve, hogy az eredeti módszertan is teljes bizonyossággal észlelte az elrejtett információt, a kiterjesztett módszertan alkalmazásától eltekintettem.

Ezek után a normaértékes LSB rejtés alkalmazása következett, 0-s szórásküszöbbel. Ez tehát azt jelenti, hogy minden olyan blokkban történik rejtés, ahol a szórás nagyobb 0-nál. A szórás márpedig pontosan akkor 0, ha a vizsgált minta minden eleme a várható értéket veszi fel, tehát a blokk tökéletesen homogén. Erre meglehetősen kicsi az esély egy fénykép esetében, így a képekben a normaértékes rejtés által felhasználható kapacitás teljes egészét kihasználtam. Az eredeti módszertan alapján elvégzett mérések azt állapították meg, hogy 98%-os bizonyossággal állítható 1 képről, hogy félig teleírták, és további 10 képről 90%-os bizonyossággal vélelmezhető, hogy az egytizedéig töltötték meg rejtett információval, valamint 9 képben a kapacitás százada került kihasználásra. Az első és az utolsó eredmény a megállapított kihasznált kapacitás szempontjából hamis pozitívnak tekinthető, bár igaz, hogy

a küszöbértékek növekedése miatt az első azt is jelenti, hogy 90%-ig bizonyos, hogy a képbe 10%-nyi adat került elrejtésre. Összességében tehát azt mondhatjuk, hogy 10 képről igen pontosan megmondta a módszertan a felhasznált szteganografikus kapacitást, és további 10 képről vélelmezhető bizonyos valószínűséggel, hogy adatok vannak bennük elrejtve.

A következő vizsgált eset a 7-es szórásküszöbbel történő adatrejtés volt. Az eredeti módszer szerinti LSB szteganalízis itt már nem volt képes hatékonyan kimutatni a rejtés tényét: mindössze 2 képről sikerült kimutatni, hogy 1%-os kihasználtsággal vannak rejtett adattal kitöltve. Érdekes, hogy pontosan ugyanaz a két hordozó „bukott le", amelyek még üres állapotukban hamis pozitívnak bizonyultak, pontosan ekkora vélelmezett kihasználtsággal. Az következik tehát mindebből, hogy az eredeti módszertan nem képes többé megbízhatóan kimutatni a rejtést, ha 7-es szórásküszöbbel végezzük azt el.

Sokkal érdekesebb viszont a kiterjesztett módszertannal kapott eredmény: mind 0-s, mind 7-es szórásküszöb mellett 100%-os vélelmezett kihasználtságot kaptam eredményül. A maximális R érték $1{,}00565$ lett, és 33 képnél volt az érték 1 alatt. Mindez azt jelenti, hogy a normaértékes LSB algoritmusra hatékony szteganalízis módszer a kiterjesztett módszertan. Az algoritmust tehát szükséges lehet továbbfejleszteni. Néhány lehetséges továbblépési lehetőséget ismertetek az 5.1. fejezetben.

4.3.5. A rejtjelezés hatása a szteganalízisre

Kíváncsi voltam, hogy a rejtjelezés módosít-e valamilyen irányban az adatrejtés szteganografikus biztonságán. A módszertan teszteléseként először a sima LSB rejtést mértem meg. Az eredmény 100% helyes pozitív, ebből 47 darab 1-nél kisebb R értékkel. A maximum arány $1{,}001092$ lett. Látható, hogy ezek a számok nem mutatnak különösebb változást a kriptográfiamentes esethez képest – mi több, megállapíthatjuk, hogy a maximum valamivel rosszabb eredményt mutat.

A következő lépésben a normaértékes rejtést mértem le, 0-s szórásküszöbbel. Az eredeti módszertan mindegyik hordozón mutatott, 1%-nyi elrejtett adatot, a ténylegesen felhasznált kapacitást azonban csak 42 esetben találta el. Érdekes, hogy bár a felismerési helyes pozitív

arány 100%, egyetlen R érték sem volt 1 alatt, és az átlag is $1{,}048706$-ra nőtt. A kiterjesztett módszertan ugyanebben az esetben a kihasznált kapacitást is 100%-osan eltalálta, és 49 esetben volt az R érték 1 alatt, ami végül az átlagot is 1 alá nyomta. Érdekes módon itt is romlás mutatkozik a rejtjelezés nélküli esethez képest.

Megvizsgáltam továbbá a 7-es szórásküszöbbel elrejtett kriptogramok esetét is. Az eredeti módszerrel végzett szteganalízis itt is érdekes eredményt hozott: 52 képben jelzett 1%-nyi kihasznált kapacitást. Bár ebben a tartományban a módszertan viszonylag megbízhatatlan (hiszen a cikkben ismeretett mérések szerint csak 10%-kal jobb az eredménye a pénzfeldobásnál), mindenképpen figyelemre méltó a megugrott találati arány a kriptográfia nélküli esethez képest. 2 képnél ezen felül a kapacitást is sikerült viszonylagosan megbecsülni. Mindazonáltal az R sosem ment 1 alá, és az átlag $1{,}133023$ lett. A kiterjesztett módszertan már jóval sikeresebb volt: 73 képben sikerült kimutatnia a teljes kihasználtságot, és az összesben észlelte az adatrejtés megtörténtét. 38 esetben ment az R érték 1 alá, és az átlag is hasonlóan alakult. A maximum $1{,}002995$ volt.

Összegezve tehát azt mondhatjuk, hogy a kriptográfia alkalmazása a felhasznált módszertannal szemben semmilyen módon nem növeli a szteganografikus biztonságot – sőt, azt is mondhatjuk a mérések alapján, hogy bizonyos esetekben még rontja is azt. Az egyetlen szignifikáns javulás, melyet a rejtjelezés behozott, a 7-es szórásküszöb mellett végzett rejtés detektálásánál látott két hamis negatív a teljes kihasználtság felismerésére nézve.

4.3.6. Kiegyensúlyozatlan forrás hatása a szteganalízisre

A mérések elvégzése során egy érdekes felfedezést tettem a stegodrive egy régebbi implementációjával. Ebben a rejtjelezés nem működött megfelelően: az első 512 bájt titkosított tartalom után csupa 0 következett, és ez az információ került végül elrejtésre. Az eredeti és a kiterjesztett módszertannal egyaránt problémákba ütköztem a szteganalízis során: a kapott R értékek szokatlanul nagyok voltak. Emlékeztetőül: még üres hordozóknál sem mértem $1{,}617955$-nél nagyobb arányt, a 0 értékű rejtett bájtok sokasága viszont több ízben is 100-as nagyságrendbe emelte az eredményt, és a 10-től 90-ig terjedő értékek voltak dominánsak.

Mindez azt jelenti, hogy ebben a szituációban ez a szteganalízis módszertan teljesen működésképtelennek bizonyult: 75 hamis negatív lett az eredmény.

A jelenség magyarázata valószínűleg az, hogy a 0-k elrejtése „foghíjassá” tette a gyakorisági táblázatot: a páratlan értékű színek majdnem teljesen eltűntek. Ez azt is jelenti, hogy egy olyan beágyazott tesztüzenet, mely nem tartalmaz szélsőségesen sok 0 értékű bájtot, drasztikusan megemeli a szomszédos színpárok számát egy rejtett információt tartalmazó hordozóban is.

Felmerül a kérdés, hogy vajon ki lehet-e használni a módszertan eme hiányosságát. Annyi bizonyos, hogy a valódi felhasználás során nem sokszor találkozunk ennyire kiegyensúlyozatlan elrejtendő adattal – ahogy az is valószínű, hogy a szteganalitikusnak rendelkezésére áll ugyanaz a statisztika, mint amely nekem is segített megmagyarázni a szteganalízis hibájtá: a gyakoriságok táblázata.

Mindazonáltal érdekes lehet megvizsgálni, ha a hordozón lenullázzuk az összes LSB-t az érdemi információrejtés előtt, akkor milyen hatékonyságot érhetünk el ezzel a módszertannal szemben. Feltételezésem szerint azzal is növelhetjük a biztonságot, hogy adunk egy olyan kölcsönösen egyértelmű transzformációt minden értékre 0-tól 255-ig, mely sok 0 értékű bittel terjeszti ki az információt, majd a transzformált adatot rejtjük el. Értelemszerűen ilyenkor a kiterjesztés arányában veszítünk a felhasználható szteganografikus kapacitásból. Egy ilyen vizsgálat esetleg képes lehet alátámasztani a rejtjelezés bekapcsolásakor tapasztalt növekedést a szteganalízis hatékonyságában.

5. fejezet

Konklúzió

Jelen dolgozat lezárásaként értékelem az eddig leírtakat, és ismertetem néhány továbbfejlesztési ötletemet.

5.1. Továbbfejlesztési lehetőségek

A javasolt algoritmusnak az alábbi továbbfejlesztési lehetőségét látom:

- A kapacitás kihasználását növelni lehetne egy adott blokk heterogeneitásának függvényében. Feltételezhető, hogy a szteganalízis módszertan esetében működésének fényében az erre a helyes megoldás, ha a blokkcentrumban levő pixel több alsó bitjét is felhasználjuk rejtésre. Ugyanis, ha több pixelben is módosítjuk az LSB-t, az potenciálisan minden egyes módosított pixelre növeli a szomszédos színpárok számát. Ha viszont egyetlen pixel több alsó bitjét használjuk, ez csak egyszer visz be potenciális extra színpárokat.

- A rejtési pozíciót álvéletlen sorozat mentén lehet érdemes megkeresni a *blokkon belül*. Ez azért lehet kedvező, mert – mint azt a 4.3.4. fejezetben már láttuk – csak a kiterjesztett módszertan volt képes megbízhatóan kimutatni az elrejtett információt; ez a lehetőség azonban megszűnik, ha a blokk bármely pixele körülbelül azonos valószínűséggel lehet rejtési pozíció, tehát a szteganalitikus kénytelen a megbízhatatlanabb eredeti módszer-

tanra hagyatkozni (vagy teljesen másik módszertan után nézni). Továbbá elmondható az is, hogy ez a kiterjesztés nem jár további kapacitásveszteséggel.

A stegodrive képességeit pedig a következőképpen lehet érdemes bővíteni:

- A már előzőleg ismertetett okok miatt drasztikus memóriaoptimalizálás szükséges.
- A felhasználtható hordozók körét bővíteni kell. A legjobb jelöltek valószínűleg a tömörített kép- és hangállományok, mert ezek meglehetősen sok számítógépen nagy számban megtalálhatóak, így jelenlétük nem kelt gyanút. Ezzel egyidejűleg célszerű az LSB-nél korszerűbb szteganográfiai algoritmusokat választani a rejtéshez. Mindezeken felül az allokációs stratégiák eszköztárát is bővíteni kell, mert a kötegelt szteganalízis veszélyeztetheti a lineáris allokációval elrejtett adatok szteganografikus biztonságát.
- A metainformációs adatbázis szteganografikus tárolását lehetővé kell tenni. Elképzelhető például egy olyan megvalósítás, melyben a felhasználó az adatbázis létrehozásakor nemcsak az adatok tárolására jelöl ki hordozókat, hanem a metainformációk számára is. Egy adatbázis megnyitásakor ez utóbbi hordozókat kellene megjelölnie a felhasználónak, és ez a halmaz – együtt a küszöbértékkel és a jelszóval – „nyitná” a tárolt metainformációs adatbázist. Ezzel könnyen biztosíthatóvá válna a hihető letagadhatóság: ha egy ellenfél kényszeríti a felhasználót, hogy magyarázza meg, miért található meg a stegodrive a merevlemezén, utóbbi egyszerűen egy másik metainformációs hordozó halmazt jelöl ki, melyek valamelyest kompromittálónak tűnő, ámde az igazi rejtett információnál kevésbé problémás adatokra mutatnak.
- A csúszka állításának nyitott adatbázis melletti lehetővé tétele biztosítaná, hogy a felhasználónak azonnal legyen információja arról, hogy mekkora kapacitáshoz jut adott csúszkaérték mellett.
- Fejlesztői szempontból érdemes lehet egységes programozói felületet (API-t) létrehozni, hogy a stegodrive könnyen bővíthető legyen újabb hordozókkal és algoritmusokkal. A

jelenlegi implementációban az algoritmusok nem „cserélhetőek”, a programkód módosítása nélkül nem lehet újakat betenni. Egy programozói felület segítségével egyszerűen „hozzáírással” lenne bővíthető a szoftver (hasonlóan például ahhoz a megvalósításhoz, amellyel a HSQLDB használja a JDBC API-t).

- Implementáció illesztőprogramban, valódi fájlrendszerként. Ez talán mind közül a legnehezebb, de a legelőremutatóbb feladat. A nehézséget az adja, hogy valószínűleg szakítani kell a Java programnyelvvel, és az egész koncepciót meg kell írni mondjuk C++ nyelven. Ezen kívül át kell gondolni, hogyan alkalmazkodunk a különböző platformok fájlrendszer-szemantikájához – Linux alá például a FUSE könyvtársegítségével lehetséges viszonylag könnyen állományrendszer-illesztőt írni (ehhez elvileg létezik Java-illesztőfelület, mindazonáltal annak stabilitása erősen megkérdőjelezhető[1]).

5.2. Értékelés

Jelen diplomatervben ismertettem koncepciómat egy újszerű szteganografikus fájlrendszerről, és javaslatot tettem egy algoritmusra, mely biztonságosabbá teheti az LSB-típusú adatrejtési módszert. Megvalósítottam egy szoftvert, mely – bizonyos értelemben vett szuboptimális volta és ma még kevés funkciója ellenére – hasznos tagja lehet mindazok eszköztárának, akik adataik biztonságát a legsokoldalúbban szeretnék megóvni.

A munka során sokféle tanulsággal lettem gazdagabb. Az első, hogy igazán biztonságos szteganográfiai algoritmust tervezni nehéz. Olyannyira így van ez, hogy magam is megtaláltam a módját a javasolt módszer észlelésének. Javaslatot is tettem azonban a feltárt hiányosságok orvoslására is, későbbre hagyva az implementációt. Mindazonáltal a szteganalízis eszköztára ezen bőven túlmutat, és sok függ attól is, hogy a támadó mire képes. Demonstráltam, hogy létezik olyan, életszerű támadó modell, melyben a szoftver megvédi a felhasználót az ellenféltől, feltéve, hogy a programban implementált, és végül felhasználásra került algoritmusok egyenként nem szteganalizálhatóak megbízhatóan.

[1] http://apps.sourceforge.net/mediawiki/fuse/index.php?title=LanguageBindings#Java

Egy másik tanulság, hogy egy szteganografikus fájlrendszer implementálása meglehetősen nagy munka. Bár örültem, hogy az eredeti koncepcióhoz nagyjából végig sikerült hű maradni, és csak apróbb kiegészítések (mint amilyen például az allokátor mint önálló feladatkörrel rendelkező objektumosztály definiálása) voltak szükségesek ahhoz, hogy a program a jelenlegi formájában működni tudjon, látom, hogy még nagyon sok hasznos dolgot kell implementálni ahhoz, hogy a szoftver igazán könnyen, sokoldalúan és megbízhatóan használható legyen. Az egyes építőkockák – adatbáziskezelés, allokációs stratégia, egy egyszerű LSB algoritmus stb. – persze önmagukban nem bonyolultak, de megfelelő együttműködésük mégis tetemes munka eredménye.

Összességében úgy gondolom, hogy a stegodrive fejlesztésére és analízisére fordított idő nagymértékben hasznos volt, és a szoftver – némi továbbfejlesztéssel – újszerű implementációja egy eddig nem elterjedt módon hasznosítja a szteganográfiát. Úgy érzem, az itt szerzett tapasztalatokat a jövőben hatékonyan tudom majd hasznosítani, és a diplomterv műszaki informatikusi tanulmányaim méltó lezárása.

Irodalomjegyzék

[1] Papapanagiotou, Kellinis, Marias, Georgiadis, *Alternatives for Multimedia Messaging System Steganography*, University of Athens, KPMG LLP

[2] Robie, Mersereau, „Video Error Correction Using Steganography”, *Journal of Applied Signal Processing*, 2002:2, pp. 164–173

[3] Eltemetett bitek: szteganográfia (2008. október 13.), https://www.netacademia.net/publikacio/cikk/pdf/0303stegano.pdf

[4] Hopper, Langford, Ahn, *Provable Secure Steganography*, School of Computer Science, Carnegie Mellon University

[5] Unicsovics György (2008. október 13.), A szteganográfia elemeinek bemutatása, http://www.hetpecset.hu/ppt_XXVIII_2/Hetpecset.pdf

[6] Cvejic, *Algorithms for Audio Watermarking and Steganography*, Oulu University Press, Oulu, 2004

[7] Anderson, Petitcolas, „On The Limits Of Steganography”, *IEEE Journal of Selected Areas in Communications*, 16, 4, pp. 474–481, 1998

[8] Chandramouli, Memon, „Security and Watermarking of Multimedia Contents” *Proceedings of the SPIE*, vol 5020, pp. 173–177 (2003).

[9] Westfeld, Wolf, *Steganography in a Video Conferencing System*, Dresden University of Technology

[10] Kerckhoffs, „La Cryptographie Militaire”, *Journal Des Sciences Militaires*, 1883:9, pp. 5–30

[11] Hartung, Girod, *Watermarking of Compressed and Uncompressed Video*, University of Erlangen-Nuremberg

[12] Danezis, *Covert Communications Despite Traffic Data Retention*, Microsoft Research

[13] „Directive 2006/24/EC of the European Parliament and of the Council”, *Official Journal of the European Union*, 105, pp. 54–63, 2006

[14] Tyler Gibson (2008. október 13.), Methods of Audio Steganography, http://www.snotmonkey.com/work/school/405/methods.html

[15] Chip Fleming (2008. október 14.), A Tutorial on Convolutional Coding with Viterbi Decoding, http://home.netcom.com/ chip.f/viterbi/tutorial.html

[16] Fridrich, Goljan, *Practical Steganalysis of Digital Images – State of the Art*, SUNY Binghamton

[17] Fehér Gábor (2008. október 14.), Adatrejtés képekben, http://qosip.tmit.bme.hu/twiki/pub/Main/InfoSzolgBizt/11-Adatrejtes_kepekben.pdf

[18] Goldberg, *A Pseudonymous Communications Infrastructure for the Internet*, UC Berkeley, 2000

[19] Yang, Huang, „A Novel Watermarking Technique for Tampering Detection in Digital Images”, *Electronic Letters on Computer Vision and Image Analysis*, 3, 1, pp. 1–12, 2004

[20] Fridrich, Du, Long, „Steganalysis of LSB Encoding in Color Images”, *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference*, New York (United States of America), 2000, vol 3., pp. 1279–1282

[21] Fridrich, Goljan, Du, „Reliable Detection of LSB Steganography in Color and Grayscale Images”, *Proc. ACM, Special Session on Multimedia Security and Watermarking*, Ottawa, Canada, 2001, pp. 27–30.

[22] Weiwei, Yanqing, Xiangwei, „JPEG Quantization-Distribution Steganalytic Method Attacking JSteg”, *IJCSNS International Journal of Computer Science and Network Security*, volt.6 No.7B, 2006, pp. 192–195

[23] Andrew D. McDonald, Markus G. Kuhn, *StegFS: A Steganographic File System for Linux*, in A. Pfitzmann (Ed.): IH'99, LNCS 1768, pp. 463–477, 2000.

[24] HSQLDB (2009. május 14.), http://hsqldb.org/

[25] Zhou, *Steganographic File System*, National University of Singapore, 2005

[26] Ker, *Perturbation Hiding and the Batch Steganography Problem*, Oxford University Computing Laboratory

[27] Hitchens, *Java NIO*, O'Reilly, 2002

Függelék – CD-melléklet

A CD-melléklet tartalma a következő:

/stegodrive/ A stegodrive jelenlegi implementációja.

/stats/ A statisztikakészítő szoftver.

/diplomaterv/ A diplomaterv elektronikus formátumban.

Az első két könyvtárban két-két alkönyvtár van:

src/ Forráskód.

bin/ Lefordított, csomagolt bináris. A futtatás a `java -jar fajlnev.jar` paranccsal lehetséges, ha a parancsértelmező keresési útvonalainak valamelyikén megtalálható a Java futtatókörnyezet.