Att dölja information i grafiska filformat

JOHANNES ELGH
och JOHAN LENNEFALK

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

Att dölja information i grafiska filformat

J O H A N N E S E L G H
o c h J O H A N L E N N E F A L K

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Lars Kjelldahl
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/elgh_johannes_OCH_lennefalk_johan_K10045.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

“Scientia potentia nostra — Semper Occultus“

Referat

Denna kandidatuppsats behandlar området steganografi och utgår ifrån problemformuleringen: *hur det är möjligt att dölja information i grafiska filformat med hjälp av den steganografiska metoden modifiering av den minst signifikanta biten.* Uppsatsen undersöker, de redan existerande, metoderna och algoritmerna för att dölja information och bidrar även med en analys av en egen alternativ algoritm. Detta tillvägagångssätt, benämnt PIGO, tillför ökad säkerhet men har i denna uppsats endast implementerats i syfte att illustrera möjligheterna med steganografi.

Abstract

To hide information in graphical file formats

This Degree Project in Computer Science discusses the area of steganography and evaluates the following question: *how is it possible to hide information in graphical file formats with the aid of the steganographic method least significant bit insertion.* The essay analyzes the, already existing, methods and algorithms available to hide information and also contributes with an analysis of an own algorithm. This procedure, named PIGO, adds increased security but has in this essay only been implemented to fulfill the purpose of illustrating the possibilities that steganography holds.

Innehåll

Figurer

Tabeller

Kapitel 1

Introduktion

1.1 Bakgrund

Under århundradenas lopp har problemet att kommunicera hemlig information alltid varit aktuellt. Vanligtvis har någon form av kryptering använts vilket också har medfört att krypterade meddelanden av sig självt genast framstått som *hemliga*. Poängen med *steganografi* är att meddelanden inte själva kan urskiljas från mängden då de framstår som *vanliga*. Själva ordet härstammar från grekiskan och översätts till *concealed writing* på engelska eller dold skrift på svenska. [3]

1.2 Problemformulering

Problemet med mycket av dagens kryptering är att de resulterande meddelandena framstår som *hemliga* och således även blir *suspekta*. Därav drar de uppmärksamhet till sig. Detta kan lösas genom att förklä meddelanden i andra skepnader. Följaktligen skall en observatör som får fri tillgång till meddelandet inte kunna avgöra om det innehåller någon hemlig information eller ej. Av denna anledning har vi bestämt oss för denna problematisering: *Hur det är möjligt att dölja information i grafiska filformat med hjälp av den steganografiska metoden modifiering av den minst signifikanta biten*. För information om denna specifika metod, se sektion 3.1 på sidan 9.

1.3 Syfte

Syftet med denna uppsats är att beskriva området steganografi i grafiska filformat och utforska några av de tillvägagångssätt som finns etablerade idag. Kortfattat skall även några algoritmer presenteras som attackerar problemet på skilda sätt och i olika format. Utöver detta är även målsättningen att implementera en egen algoritm i ett relativt enkelt format.

Del I

Bakomliggande teori & fakta

Kapitel 2

Allmän teori

2.1 Byte och bitar

En bit är den minsta informationsbärande enheten i en dator. Denna kan anta noll eller ett; det vill säga på eller av. Om två bitar används tillsammans finns $2^2 = 4$ kombinationer 00, 01, 10 och 11. Det är med kombinationer av ettor och nollor som all data beskrivs. Åtta bitar tillsammans bildar en *byte* och kan representera talen 0 till 255 ($2^8 = 256$). Med dessa 256 kombinationer kan bokstäver, siffror och specialtecken beskrivas.

2.2 BMP

Bildformatet BMP, som är en förkortning av det engelska ordet bitmap, är ett brett använt filformat på Microsoft Windows-operativsystem. Bildformatet är okomprimerat, lagrat i matrisform och använder sig av färgblandningsmetoden RGB. BMP-filer är uppbyggda enligt följande struktur: först kommer ett relativt kort filhuvud (eng. *header*) och därefter följer en matris som representerar samtliga pixlar i bilden. I filhuvudet ligger bland annat information som beskriver dess höjd, bredd och bitdjup. Då alla pixlar har en egen plats i matrisen betyder detta att BMP-filer ofta blir mycket stora. Denna enkelhet är dock det som gör det till ett lämpligt filformat att illustrera steganografi med. [4] [7]

2.3 Hexadecimalt

Det hexadecimala talbassystemet används ofta när bitar skall beskrivas för att slippa skriva ut alla nollor och ettor. Detta talbassytem bygger på basen 16 istället för basen 10 som vi är vana vid. Siffrorna 0-9 beskrivs som vanligt medan talen 10 till 15 i det decimala talbassystemet representeras av bokstäverna A-F i det hexadecimala. För att skriva talet 15 hexadecimalt skrivs alltså enbart F medan 16_{10} skrivs som 10_{16}. Vanligtvis skrivs den använda basen ut nedsänkt till höger om det specifika talet för att beskriva vilket numeriskt talbassystem talet befinner sig i.

Det lämpar sig bättre att beskriva bitar och/eller byte i det hexadecimala systemet än i det decimala. Detta beror på att fyra bitar ($2^4 = 16$ kombinationer) kan beskrivas av ett tecken (0-9, A-F) i detta talbassystem. Således kan en byte, åtta bitar, beskrivas av två stycken hexadecimala tecken.

2.4 JPEG-komprimering

Ett av de bildkomprimeringsformat som ofta används idag är JPEG. Förkortningen står för *Joint Photographic Experts Group* och syftar på en standard för destruktiv bildkomprimering. Algoritmen bygger på diskreta cosinustransformer (DCT) som utförs i 8×8-block och delas in i 64 DCT-koefficienter vardera [11]. Utformningen lämpar sig bra för fotografiska bilder då den är gjord så att man utnyttjar ögats oförmåga att exakt urskilja skillnader i färggradienter. Således fungerar algoritmen väl för människor men kan t.ex. vid maskininläsning vara direkt olämplig. Vid komprimering av artificiella bilder, som ritningar eller dylikt där räta linjer är ett vanligt förekommande element, är därför denna komprimeringsstandarden ej att rekommendera. Detta då komprimeringen i dessa fall ofta blir allt för framträdande om en bra komprimeringsgrad skall uppnås. Ett sätt att använda JPEG-komprimerade bilder som ett medium för steganografi är att modifiera de minst signifikanta bitarna hos de normaliserade DCT-koefficienterna. Detta är dock möjligt att genomskåda med hjälp av statistisk analys. Således är detta ett relativt avancerat men samtidigt osäkert sätt att tillämpa steganografi på. Därav blir det ur ett pedagogiskt synsätt betydligt lättare att illustrera användandet av steganografi med något annat format. [5] [12]

2.5 PNG-komprimering

PNG-komprimering är till skillnad från JPEG-komprimering en icke-förstörande komprimering. PNG-formatet är ett palettbaserat bildformat där, istället för att definiera varje pixel var och en för sig, samtliga färger fördefinieras och varje pixel sedan får ett index som färgerna fördelas till. Således fungerar denna komprimering bäst på bilder med stora enfärgade områden då paletten annars blir stor om många färger skall definieras däri. [6]

2.6 RGB

RGB är en additiv färgblandningsmetod för att kunna representera olika färger. Genom att lägga ihop kombinationer av olika mängder Röd, Grön och Blå (eng. *Red, Green, Blue*) återfås en mängd olika färger. Vanligtvis används 24 bitar eller åtta bitar för att representera en färg i en digital bild. Användandet av den förstnämnda standarden kan resultera i väldigt stora bilder och således bidra till mycket extra utrymme att dölja information i. I lagringsformatet 24 bitar används tre byte för att beskriva en pixel; en byte för var och en av de tre primärfärgerna: röd, grön och

blå. För att få en svart eller vit pixel nollställs respektive ettställs samtliga bitar i pixeln.

Istället för att beskriva en pixel med tre gånger åtta stycken ettor/nollor brukar dessa skrivas hexadecimalt. Det blir då tre gånger två tecken.

Kapitel 3

Steganografiska algoritmer

3.1 Modifiering av den minst signifikanta biten

Minst-signifikanta-bit-insättningsmetoden (eng. *least significant bit insertion LSB*) är en relativt simpel och välanvänd steganografisk teknik [10]. Då 24-bitars färgrepresentation används kan 16,777,216 olika färger bildas och således finns det stora möjligheter att dölja information däri [9]. Namnet på tekniken är självförklarande; metoden går ut på att gömma information i den minst signifikanta biten i bildens alla byte [13]. I en bild om 600×600 pixlar (i 24-bitars färgrepresentation; tre byte per pixel) finns då möjligheten att gömma $600 \cdot 600 \cdot 3 = 1,080,000$ bitar eller $\frac{600 \cdot 600 \cdot 3}{8} = 135,000$ byte [10]. Om mycket information måste döljas finns möjligheten att använda de två minst signifikanta bitarna i varje byte [10]. Trivialt kan då inses att den dubbla mängden information är möjlig att dölja gentemot ifall bara den minst signifikanta biten används.

Det självklara problemet med denna teknik är att bildens originalfärger *ej* bi-

(a) Originalbilden **(b)** Bild med hemligt meddelande

Figur 3.1: Två stycken 20×20 pixlar stora bilder (uppskalade). Ursprungligen är de båda bilderna lika. Nu är dock meddelandet "HEJ" gömt uppe i högra hörnet på den högra bilden

behålls. Skillnaden eskalerar med kvadraten för varje extra bit som används för att dölja information.

Om de två minst signifikanta bitarna i varje byte används för att skriva ett meddelande och observatören känner till att bilden blivit steganografiskt behandlad och vet vilka originalfärgerna är, finns möjlighet för denna observatör att upptäcka detta. Om emellertid detta inte är fallet, är risken att det mänskliga ögat skulle uppfatta något suspekt med bilden mycket liten [10].

Följande exempel illustrerar hur bokstaven S göms i tre godtyckliga pixlar. S har det hexadecimala värdet 53_{16} enligt ASCII-standarden och representeras binärt 01010011_2. Detta binära tal delas upp i enskilda bitar som sedan ersätter den minst signifikanta biten i varje byte hos ursprungsdata. Nedan följer tre godtyckliga pixlar av en bild i 24-bitars färgrepresentation (en pixel är tre byte):

01011001	01011001	11111010
01001000	11110010	10110011
01100110	01001100	01011111

Nedan ersätts sista biten i de åtta första byten för att dölja bokstaven S (01010011_2). Detta har gjorts radvis från vänster till höger.

0101100**0**	0101100**1**	1111101**0**
0100100**1**	1111001**0**	1011001**0**
0110011**1**	0100110**1**	01011111

En vit pixel beskrivs, som tidigare nämnts, av tre fullt ettställda byte; vilket hexadecimalt skulle skrivas $FF_{16}FF_{16}FF_{16}$ (det vill säga $255_{10}255_{10}255_{10}$). Om de två sista bitarna i varje färg används för att gömma ett meddelande kan den största skillnaden mot den vita färgen uppstå då alla dessa två gånger tre bitar nollställs. Figur 3.2 illustrerar den maximala skillnaden som kan uppnås mot helt

Figur 3.2: I figuren, inom den svarta ramen, finns två lika stora kvadrater sida vid sida. Till vänster, helt vit färg (hexadecimal färgkod: FFFFFF), till höger nästan vit färg (hexadecimal färgkod: FCFCFC).

vit färg. Skillnaden syns bäst på en datorskärm med något sämre betraktningsvinkel; exempelvis en laptop.

3.2 LSB med palettbaserade bildformat

Problemet med BMP är att filerna blir så stora lagringsmässigt. Alla pixlar definieras med tre byte, en för varje färg. Det är dock högst troligt att närliggande pixlar i en bild har samma färgkod. Det är då onödigt att återdefiniera dessa tre byte om och om igen. Ett palettbaserat bildformat är en typ av bitmap där, istället för att definiera varje pixel var och en för sig, samtliga färger fördefinieras och varje pixel sedan får ett index som färgerna fördelas till. Denna palett kan utformas på många olika sätt där ett vanligt förekommande sådant är efter färgernas användningsfrekvens. Detta för att reducera medeltiden för att hitta en indexerad färg. [10]

Palettbaserade bildformat är mycket mer använda på Internet på grund av deras mindre storlek [10]. Exempel på dessa bildformat är GIF och PNG. Det finns dock andra skillnader. GIF använder till exempel enbart en byte för att beskriva en färg. Detta ger att det bara finns $2^8 = 256$ olika färger att välja mellan. Detta reducerar storleken på filen mycket, dock blir *kvaliteten* sämre. Ur ett steganografiskt perspektiv är detta sämre. Med färre byte finns inte lika mycket utrymma att gömma data i. Dessutom, med bara 256 olika färger blir skillnaden *relativt* stor då den minst signifikanta biten ändras. [10]

I och med det ovan nämnda bör extra försiktighet och noggrannhet tas när MSB-tekniken används på palettbaserade bildformat. En ändring i den minst signifikanta biten kan generera en helt ny färg. Detta beror på att, som ovan nämnt, färgerna inte behöver vara sorterade efter färgspektrumet utan oftast är sorterade efter vilken färg som används mest. En möjlig lösning på detta är att sortera om paletten efter färgspektrumet så att skillnaden blir minimal. En annan lösning skulle kunna vara att lägga till nya, visuellt närliggande, färger till de redan existerande. Problem kan då uppstå om antalet unika färger i originalbilden understiger det maximala antalet färger (detta beror av bitdjupet). Följaktligen blir valet av originalbilden avgörande. Om en gråskalig bild väljs finns det 256 nyanser av grått och således blir skillnaden mellan två intilliggande nyanser liten och svårupptäckt. [10]

3.3 Fingerprint & Watermark

I dagens samhälle finns det många användningsområden för steganografi. Då digital distribution av olika media blir allt mer förekommande ökar även den illegala spridningen av dessa. Det finns således ett behov av att märka och kunna identifiera bilder, filmer eller musik som har kopierats. Vanligtvis används två olika metoder: watermarking och fingerprinting. I det förstnämnda fallet handlar det om att vattenmärka en fil så att den blir identifierbar i efterhand. Det kan t.ex. handla om att en fil som har sparats i ett visst program bär en *osynlig* signatur. Ordet kommer

ur uttrycket att vattenmärka vilket syftar till det sätt som sedlar, pass och andra värdepapper ofta märks. I det andra fallet, fingerprinting, handlar det oftast om att inkludera specifik information såsom någon form av licensnummer i filen. Ordet refererar till det unika fingeravtryck som vi alla bär med oss. På detta sätt är det möjligt att spåra vissa data till en specifik användare. Detta används bl.a. i skrivare där varje utskriven kopia får ett unikt ID-nummer. Om användaren således har registrerat sin skrivare är det möjligt att spåra upp vem som har skrivit ut ett visst dokument. I båda dessa fall handlar det om att gömma information på ett sådant sätt att den inte enkelt upptäcks.

3.4 Att dölja information på smartare sätt

Den simplaste formen av steganografi, där ett meddelande i sin enkelhet bäddas in i följd bland de minst signifikanta bitarna i bilden (även kallad BlindHide), är väldigt osäker och lättavslöjad. För att undvika att steganografi i bilder skall upptäckas har därför flera algoritmer utvecklats. Det finns många varianter på dessa och de försöker på olika sätt att intelligent dölja information i bildformat. Nedan presenteras tre olika sätt att göra detta på.

3.4.1 HideSeek

Denna algoritm utnyttjar ett lösenord som sedan omvandlas till en *hash* och sedan används som *frö* för att generera en serie slumptal. Detta betyder att ett lösenord, via en algoritm, bryts ner till ett tal. Detta tal används sedan som utgångspunkt för att generera denna serie av slumptal. Ett visst frö kommer alltid att generera samma serie och således är det enda sättet att återskapa dessa tal att känna till lösenordet. Slumptalen används sedan för att placera ut var i bilden det hemliga meddelandet skall placeras. Därav är denna algoritm betydligt säkrare än att placera meddelandet i en serie av efterföljande pixlar. Svagheten ligger dock i det fall där meddelandet skall bäddas in i bild med bara ett fåtal färger; som t.ex. en bild på den Svenska flaggan. I detta fall skulle således de modifierade pixlarna, som representerar det hemliga meddelandet, relativt enkelt framträda för det blotta ögat. Genom att via programvara t.ex. öka kontrasten skulle dessa pixlar enkelt kunna identifieras. [8]

3.4.2 FilterFirst

Både BlindHide och HideSeek är algoritmer som relativt lätt kan genomskådas då ett block av modifierade pixlar enkelt kan urskiljas mot omkringliggande enhetliga färguppsättning. Av denna anledning finns flertalet lösningar på detta problem där en av algoritmerna som syftar till att lösa detta går under namnet *FilterFirst*. Tanken ligger i att först detektera kanter i bilden genom att t.ex. använda Laplacetransformer. Således identifieras områden i bilden där omkringliggande pixlar skiljer sig så mycket som möjligt. Följaktligen döljs sedan den hemliga informationen i dessa pixlar. Genom att bara alternera de x minst signifikanta pixlarna och således

använda de y mest signifikanta bitarna för att filtrera där $1 \leq x \leq 7$ och $y = 8 - x$ kommer alltid detta filter ge samma resultat, både före och efter att information har dolts. Således krävs ingen ytterligare information för denna typ av algoritm. Nackdelen hos denna algoritm ligger istället i säkerheten. Genom att enkelt upprepa filtreringen kan de alternerade pixlarna extraheras ur bilden. Den uppenbara fördelen framgår av samma skäl; det är enkelt att utvinna information ur en bild. [8]

3.4.3 BattleSteg

För att råda bot på svagheterna hos de ovanstående algoritmerna utvecklade Kathryn Hempstalk på University of Waikato en ny algoritm som en kombination av de båda; *BattleSteg*. Förkortningen kommer ur orden *Battleships Steganography* som på engelska syftar på det klassiska brädspelet "Sänka skepp" (eng. *Battleships*). Algoritmen väljer ut de h % bästa platserna att gömma information på och benämner dessa områden *skepp*. Sedan placeras *skott* ut slumpvis i bilden till dess att ett skepp träffas. När denna träff sker placeras sedan de kommande skotten i ett kluster kring denna position i hopp om att träffa fler skepp. Efter i antal skott återgår algoritmen till att välja pixlar slumvalt för att undvika att stora kluster med pixlar hamnar allt för nära varandra och således enklare upptäcks i bilden. Således undviker BattleSteg att stora block med pixlar innehållande information hamnar i närheten på ett bättre sätt än HideSeek. Säkerheten ligger dessutom på en liknande nivå då det är omöjligt att placera "skotten" på samma sätt om inte fröet är känt som genererar denna serie med slumptal. [8]

3.4.4 Analys av dessa algoritmer

För att analysera dessa algoritmer har Kathryn Hempstalk valt att använda flertalet avancerade algoritmer som inte ämnas behandla här. Slutsatsen var dock att FilterFirst var betydligt mer effektiv på att dölja information än de två traditionella algoritmerna HideSeek och BlindHide. BattleSteg var dock endast bättre än HideSeek och BlindHide hamnade mellan BattleSteg och FilterFirst. Det är dock inte rekommenderat att använda BlindHide på grund av dess dåliga säkerhet och möjligheten att upptäcka hemliga meddelanden i bilder med blotta ögat. Således är FilterFirst den mest lämpade algoritmen av dessa då den både är relativt säker och svår att upptäcka.

3.5 Att upptäcka steganografi

Det finns flera olika sätt att upptäcka användandet av steganografi i bilder på okrypterad form. Vid modifiering av de minst signifikanta bitarna i BMP-bilder uppstår ett brus i bilden vilket leder till att bilden kan verka suspekt. Vid modifiering av DCT-koefficienterna i JPEG-formatet förändras dessutom histogrammet som representerar fördelningen av färger i bilden. Genom statistisk analys kan denna avvikelse

upptäckas och således bilden flaggas som suspekt även i detta fall [12]. Att sedan applicera en attack i form av råstyrka (eng. *brute force*) i kombination med en lexikal analys kan i många fall leda fram till att det gömda meddelandet upptäcks [12].

Del II

Egen undersökning

Kapitel 4

Picture-In-Garbage-Out (PIGO)

4.1 Algoritmbeskrivning

En del av syftet med denna uppsats var att uppfinna en ny algoritm for att dölja meddelanden i en bild. För att hålla nivån relativt låg har således formatet BMP valts. Detta format bygger på en matris som representerar samtliga pixlar, utan komprimering eller palett, och följaktligen är det relativt enkelt att arbeta med. Det är dessutom bättre ur ett pedagogiskt synsätt då det lättare att förklara på ett intuitivt sätt.

Picture-In-Garbage-Out (PIGO) bygger på att data döljs i pixlarna hos en bild med hjälp av slumpen och ett lösenord. Avsiktligt utelämnas djupare detaljer gällande algoritmen. Algoritmens namn syftar, inte helt oväntat, på det faktum att originalbilden smutsas ner (eng. *garbage*) för att försvåra upptäckt. Detta sker dock på ett så subtilt sätt att det lika gärna skulle kunna vara fråga om en dåligt komprimerad bild för blotta ögat. Således blir det mycket svårt för en analytisk algoritm att hitta det gömda meddelandet.

4.2 Metod

För att utvärdera effektiviteten hos PIGO implementerades algoritmen och utfallen av olika indata mättes. Dessa resultat har sedan plottats i grafer för att illustrera hur de olika utfallen beror av flertalet faktorer såsom mängden data att dölja i förhållande till bildstorleken. För att få tag i en stor mängd data att maskera inuti en bild valdes den engelska versionen av King James Bibel (ca 4.8 MB ren text) [1]. Till detta krävdes även ett stort medium i form av en bild att dölja informationen i. Genom att använda en högupplöst bild med över 14 miljoner pixlar föreställande fyra grafer [2] löstes detta. För att även studera, det kanske mer realistiska, fallet där data döljs inuti ett fotografi togs ett sådant föreställande de två författarna till denna uppsats, Johan Lennefalk och Johannes Elgh. Då denna bild fångades med hjälp av webbkameran till en Apple MacBook Pro är upplösningen betydligt lägre, omkring 300,000 pixlar. På detta sätt representerar dessa bilder två olika kategorier

av bilder som cirkulerar på Internet, fotografier och datorgenererad grafik; således är de mycket lämpliga att dölja meddelanden i. Då storleken på dem dessutom varierar mycket gör de analysen mer omfattande och mångfacetterad.

Båda försöken utfördes på en, vid tillfället, modern bärbar dator av modell Apple MacBook Pro med en 2.66 GHz Intel Core 2 Duo CPU och 4 GB 1067 MHz DDR3 RAM-minne.

4.3 Implementering av algoritmen

Vi valde att implementera algoritmen med Java av enkelhets skäl. Själva algoritmen tar en pixelmatris, ett lösenord och meddelandet att gömma som indata. Detta gör att algoritmen måste bäddas in i ett större program som tar en BMP-bild snarare än en pixelmatris som indata. Det första programmet gör är att läsa in hela BMP-bilden som en byte-vektor och sedan placera in varje pixel i en $N \times M$-matris (där $N \times M$ motsvarar bildens dimension). Denna matris används sedan som indata till algoritmen. Då vi utgår från den steganografiska metoden *modifiering av den minst signifikanta biten*, och således hela byte kan läggas in, har meddelandets format ingen betydelse. Således finns möjlighet att gömma till exempel en bild, en text eller en ljudfil. Den enda begränsningen är storleken på det som skall gömmas.

Som tidigare beskrivits krävs det tre pixlar för att gömma en byte. För att representera tre pixlar med formatet RGB och 24 bitars färgdjup krävs nio byte, och dessa nio har varsin minst-signifikanta-bit tillgänglig att kunna gömma en bit i. Den sista av dessa nio bitar lämnas oförändrad (en byte är åtta bitar). Detta ger slutligen att en bild med upplösningen 640×480 pixlar, det vill säga i totalt $640 \cdot 480 = 307,200$ pixlar, kan gömma $\frac{640 \cdot 480}{3} = 102,400$ byte (102 kB). Mängden data, som i detta fall utgörs av 102,400 byte, har vi valt att benämna *maximal meddelandelängd*.

Som nämndes i beskrivningen av algoritmen utelämnas här en djupare beskrivning av denna. Då algoritmen har genomförts, återfås samma pixelmatris, dock en aning modifierad. När sedan pixelmatrisen återtransformerats till en byte-vektor används samma header-data som inkom med originalbilden för att åter igen skriva ned dessa till en bildfil.

4.4 Resultat

I de kommande sektionerna presenteras data och grafer som illustrerar utfallen av de två testscenarion som tidigare beskrivits i sektionen *Metod* (se sektion 4.2). I de båda fallen har elva olika försök gjorts med varierande mängd data inbäddad i bilden för att se hur exekveringstid av algoritmen, PIGO, beror av detta. Dessutom har de resulterande bilderna komprimerats med hjälp av det icke-förstörande bildformatet PNG för att se huruvida dessa skulle kunna spridas på Internet utan att framstå som suspekta. I uppsatsens nästa kapitel följer en sektion där slutsatser utifrån detta dras och en kort analys ges.

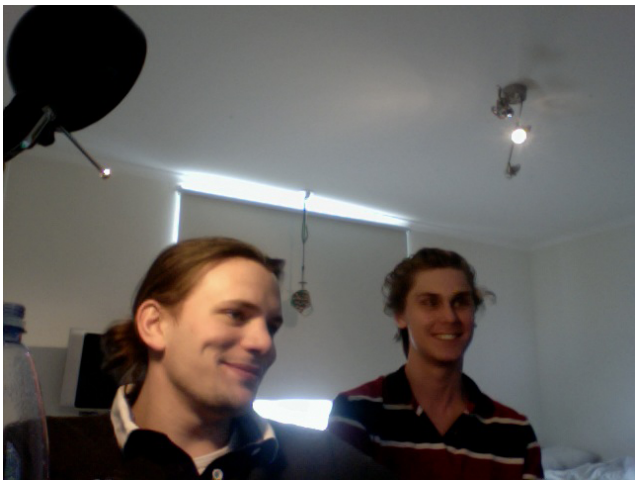
Viktigt att notera är att PIGO *inte* ändrar bildstorleken på BMP-filen; däremot förändras färgerna och nya färger kan uppstå. Detta medför, i många fall, att PNG-komprimering inte kan appliceras lika effektivt som på en orörd bild.

4.4.1 Applicering på fotografi

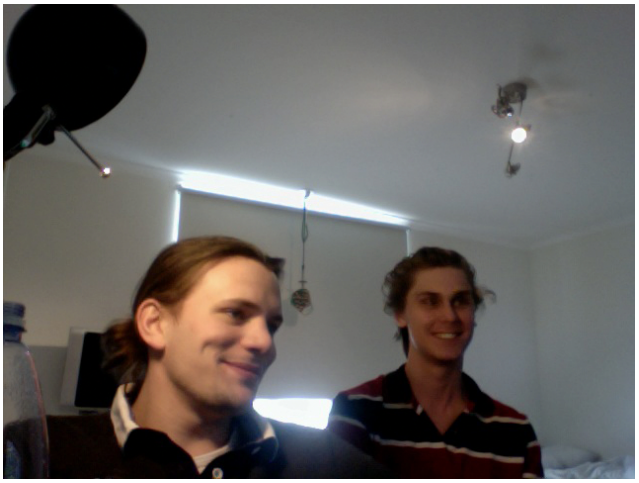
Figur 4.1a representerar en godtycklig bild som skulle kunna hittas på Internet och användas som medium för steganografi. I figur 4.1b har den förstnämnda bilden (figur 4.1a) använts som grund för att dölja information. Denna information utgörs av första och andra mosebok (eng. *Genesis* och *Exodus* [1])

Mätdata vid applicering av PIGO på fotografi

För att studera tidskomplexiteten hos algoritmen vid applicering på ett typiskt fotografi har en varierande delmängd av bibeln [1], med figur 4.1a som medium, bäddats in. Tabell 4.1 visar specifika data om figur 4.1a.



(a) Detta är originalbilden som användes vid analys av PIGO då denna algoritm appliceras på ett fotografi.



(b) Denna bild innehåller, förutom två datateknologer, första och andra mosebok (eng. *Genesis* och *Exodus* [1])

Figur 4.1: Två bilder föreställande samma fotografi före och efter bearbetning med PIGO.

Tabell 4.2 innehåller resultat från elva olika exekveringar av PIGO där olika mängder data har bäddats in med figur 4.1a som medium. Kolumnen *utnyttjad kapacitetskvot* (u_{cq}) representerar hur stor andel data i förhållande till den *maximala meddelandelängden* (se sektion 4.3) som har bäddats in i bilden. Denna kvot blir således 1.0 då varje minst signifikanta bit i varje färgkomponent (dvs. varje byte) i varje pixel har använts för att lagra data.
Kolumnen tid (t) förklarar, inte helt oväntat, körningstiden för denna specifika exekvering. Den sista kolumnen, PNG-storlek, visar hur stor bilden, innehållande meddelandet, blir efter komprimering till PNG-format.

För att lättare illustrera utfallet av försöket har vi valt att plotta de två första kolumnerna i tabell 4.2 som en graf i figur 4.2a. En exponentiell regression utfördes sedan för att undersöka hur exekveringstiden beror av kvoten. Då R^2-värdet blev 0.926 för regressionen kan denna tolkas som välanpassad. Regressionen ses som en smal svart linje i grafen och kan matematiskt uttryckas på formen nedan:

$$t = 38.21e^{2.142u_{cq}} \tag{4.1}$$

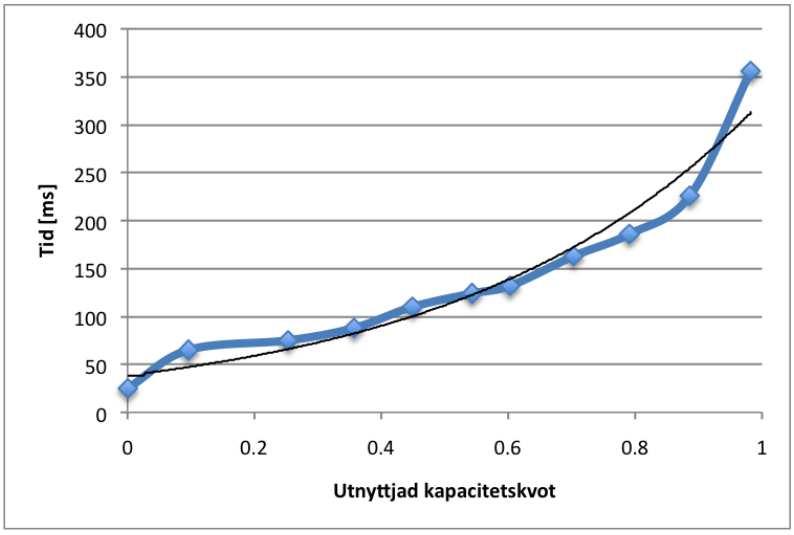
Vi valde att även plotta bildstorleken, innehållande meddelandet, efter PNG-komprimering som funktion av den utnyttjade kapacitetskvoten. Som ses i figur 4.2b ändras storleken relativt lite jämfört med originalbildens storlek efter PNG-komprimering.

Allmän bildspecifikation för fotofrafi	
Bildstorlek (BMP)	1,034,722 byte
Bilddimensioner	640×480 pixlar
Bildstorlek efter PNG-komprimering	292,750 byte

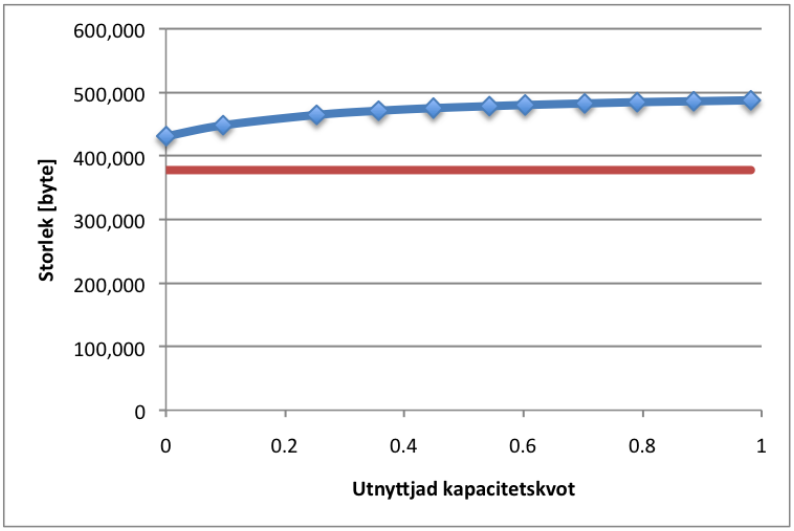
Tabell 4.1: Bilddata för Figur 4.1a

Utnyttjad kapacitetskvot (u_{cq})	Tid (t) [ms]	PNG-storlek [byte]
0.982	356	487,496
0.886	226	485,990
0.791	186	484,514
0.703	163	482,541
0.603	132	480,085
0.543	124	478,280
0.449	110	475,196
0.357	88	471,086
0.253	75	464,567
0.096	65	448,335
0.00045	25	431,175

Tabell 4.2: Resultat från de elva exekveringarna med figur 4.1a som utgångsbild.



(a) Tid plottat som funktion av den utnyttjade kapacitetskvoten. Data given i tabell 4.2



(b) PNG-komprimerad bildstorlek som funktion av den utnyttjade kapacitetskvoten. Den övre linjen representerar de elva exekveringarna med olika mängd data. Den undre konstanta linjen är storleken på den PNG-komprimerade originalbilden 4.1a. Data given i tabell 4.2 och tabell 4.1.

Figur 4.2: Två grafer representerande resultaten från försöket med applicering på fotografi

4.4.2 Applicering på datorgenererad figur

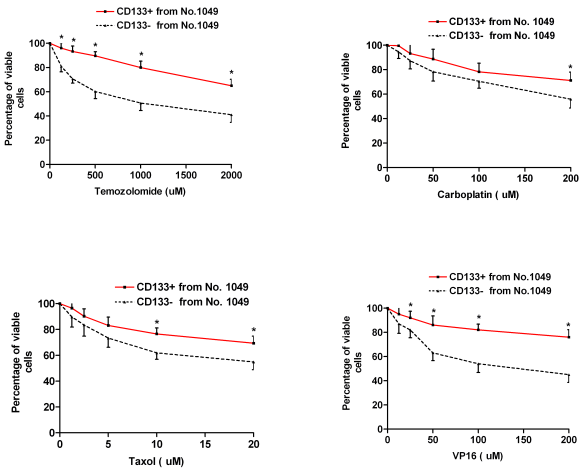
Figur 4.3a representerar en godtycklig datorgenererad figur och på samma sätt som fotografiet i sektion 4.4.1 kan denna bild användas som medium för steganografi. Bilden valdes då den är mycket stor (44.5 MB) och det såldes är möjligt att gömma mycket information i den. I figur 4.3b har den förstnämnda bilden (figur

4.3a) använts som grund för att dölja en relativt stor mängd information. Närmare bestämt—hela bibeln.

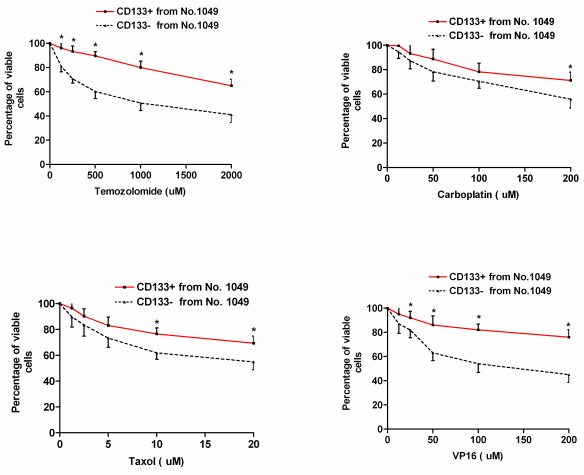
Mätdata vid applicering av PIGO på datorgenererad figur

För att studera tidskomplexiteten hos algoritmen vid applicering på ett typiskt fotografi har en varierande delmängd av bibeln [1], med figur 4.3a som medium, bäddats in. Tabell 4.3 visar specifika data om figur 4.3a. Tabell 4.4 innehåller resultat från elva olika exekveringar av PIGO där olika mängder data har bäddats in med figur 4.3a som medium. Kolumnen utnyttjad kapacitetskvot (u_{cq}) representerar hur stor andel data i förhållande till denmaximala meddelandelängden (se sektion 4.3) som har bäddats in i bilden. Kvoten blir således 1.0 då varje minst signifikanta bit i varje färgkomponent (dvs. varje byte) i varje pixel har använts för att lagra data.
Kolumnen tid (t) förklarar körningstiden för denna specifika exekvering. Den sista kolumnen, PNG-storlek, visar hur stor bilden, innehållande meddelandet, blir efter komprimering till PNG-format.

För att lättare illustrera utfallet av försöket har vi valt att plotta de två första kolumnerna i tabell 4.4 som en graf i figur 4.4a. En exponentiell regression utfördes sedan för att undersöka hur exekveringstiden beror av den utnyttjade kapacitetskvoten. Då R^2-värdet blev 0.979 för regressionen kan den-



(a) Detta är originalbilden som användes vid analys av PIGO då denna algoritm applicerades på en datorgenererad figur. För källa se [2].



(b) Hela bibeln är inbäddad i denna bild.

Figur 4.3: Två bilder föreställande samma datorgenererade figur före och efter bearbetning med PIGO. Notera att diagrammen i figuren inte har något med denna uppsats att göra utöver det faktum att de används som medium att dölja information i.

Allmän bildspecifikation för datorgenererad figur	
Bildstorlek (BMP)	44,552,442 byte
Bilddimensioner	4312×3438 pixlar
Bildstorlek efter PNG-komprimering	292,750 byte

Tabell 4.3: Bilddata för Figur 4.3a

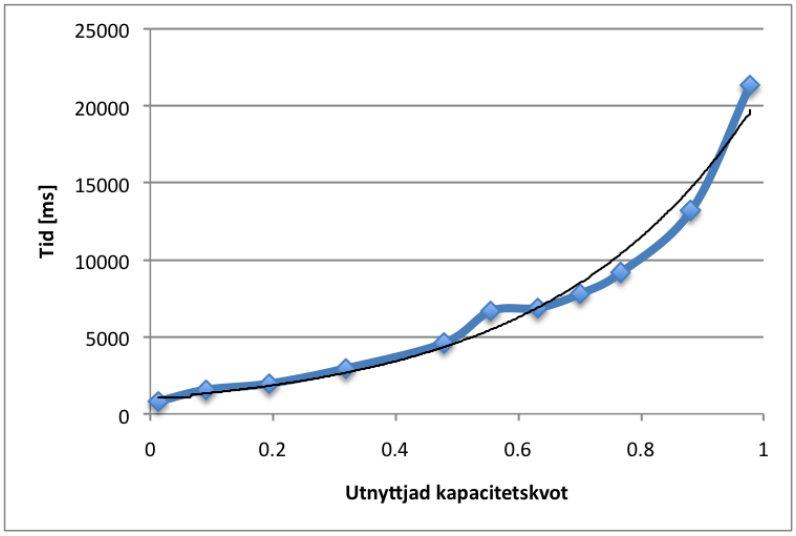
Utnyttjad kapacitetskvot (u_{cq})	Tid (t) [ms]	PNG-storlek [byte]
0.978	21,345	9,068,524
0.881	13,209	9,386,700
0.767	9,183	9,522,494
0.701	7,798	9,523,053
0.632	6,872	9,493,227
0.555	6,676	9,409,821
0.479	4,598	9,248,230
0.319	2,941	8,649,743
0.194	1,954	7,738,116
0.091	1,554	6,446,475
0.0132	802	4,713,469

Tabell 4.4: Resultat från de elva exekveringarna med figur 4.3a som utgångsbild

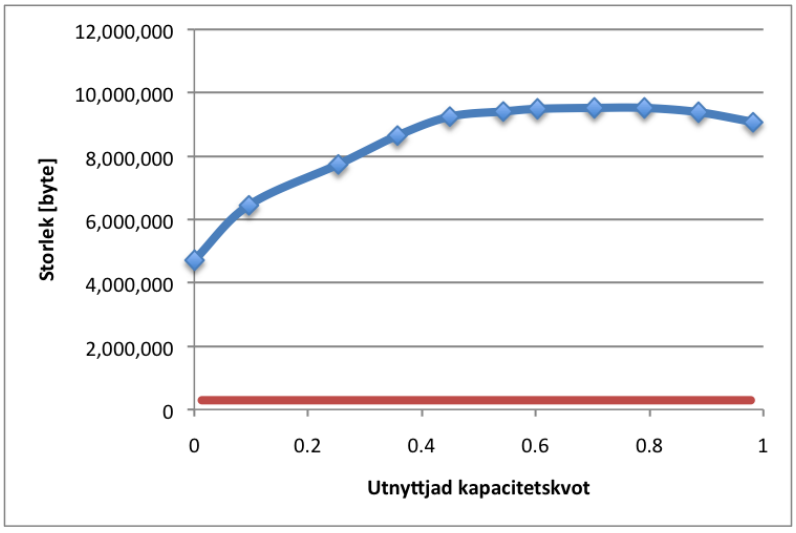
na tolkas som välanpassad. Regressionen ses som en smal svart linje i grafen och kan matematiskt uttryckas på formen nedan:

$$t = 1024e^{3.074u_{cq}} \tag{4.2}$$

Vi valde att även plotta bildstorleken, innehållande meddelandet, efter PNG-komprimering som funktion av den utnyttjade kapacitetskvoten. Som ses i figur 4.4b ändras storleken markant jämfört med originalbildens storlek efter PNG-komprimering.



(a) Tid plottat som funktion av den utnyttjade kapacitetskvoten. Data given i tabell 4.4



(b) PNG-komprimerad bildstorlek som funktion av den utnyttjade kapacitetskvoten. Den övre linjen representerar de elva exekveringarna med olika mängd data. Den undre konstanta linjen är storleken på den PNG-komprimerade originalbilden 4.3a. Data given i tabell 4.4 och tabell 4.3.

Figur 4.4: Två grafer representerande resultaten från försöket med applicering på en datorgenererad figur.

Del III

Diskussion

Kapitel 5

Analys av den egna undersökningen

5.1 Komplexitetsanalys

Då PIGO är en slumpbaserad algoritm är dess tidskomplexitet relativt svår att analysera. Det som dock framgår av figur 4.2a och 4.4a är att en utnyttjad kapacitetskvot över approximativt 0.5 ger en betydligt snabbare tidsökning. För låga värden på denna kvot kan PIGO:s tidskomplexitet beskrivas som näst intill linjär.

Slutsatsen som kan dras från de två försöken (fotografi respektive datorgenererad figur) är att mätvärdena uppvisar liknande karaktär där denna kan beskrivas som exponentiell (se ekv. (4.1) och ekv. (4.2)).

I realiteten resulterar dock inte denna tidskomplexitet i några problem då ett fåtal sekunder troligen inte påverkar det huvudsakliga syftet med algoritmen: *att dölja information i grafiska filformat.*

5.2 Säkerhetsanalys

Säkerheten hos PIGO är högre än i många andra algoritmer som bygger på *modifiering av den minst signifikanta biten.* Detta beror av att algoritmen delvis är baserad på slumpen. Resultatet av detta blir att råstyrkeattacker troligtvis ger dåligt resultat då dessa ofta bygger på att det gömda meddelandet ligger enligt en förutbestämd struktur i bilden. Utöver detta ger vår *nedsmutsning* av bilden en ökad säkerhet då det är mycket svårt att särskilja data från *smuts.*

I många fall är det möjligt att fastställa var data har gömts om tillgångs ges till både originalbild och den steganografiskt bearbetade bilden. Detta då en trivial analys kan fastställa vilka pixlar som har alternerats och på vilket sätt. I fallet med PIGO skulle dock denna typ av analys inte ge någon vidare relevant information på grund av den inbäddade *smutsen.*

Denna egenskap hos algoritmen blir även dess akilleshäl då en steganografiskt bearbetad bild ger upphov till ett betydligt större spektrum färger än den ursprungliga. Vid statistisk analys kan denna avvikelse upptäckas och bilden således flaggas som suspekt.

När ett så mycket större spektrum av färger uppkommer, kan den icke-förstörande PNG-komprimeringen ej appliceras effektivt. Detta framgår av skillnaden mellan den övre och undre linjen i figur 4.2b respektive figur 4.4b. Den undre linjen representerar PNG-storleken hos originalbilden utan meddelande i respektive fall. Tydlig skillnad ses då originalbilden inte innehåller en naturlig mängd olika färger. Som bekant existerar det i naturen (fallet fotografi) betydligt fler än tre färger (fallet datorgenererad figur). För att minimera det antal nya färger som PIGO tillför bör således en bild innehållandes ett naturligt brett färgspektrum väljas.

Kapitel 6

Slutsats

Då syftet med denna uppsats dels var att undersöka, de redan existerande, metoderna och algoritmerna för att dölja information med hjälp av steganografi men även att bidra med något nytt till denna scen kan uppsatsens omfång tyckas en aning stort och spretigt. För att råda bot på detta har ett analytiskt tillvägagångssätt tillämpats där både existerande information, ny kunskap och innovativt tänkande har applicerats. Genom att grundligt undersöka olika metoder som redan finns beskrivna i flertalet avhandlingar skapades först en bild av nuläget. Detta användes sedan som utgångspunkt för skapandet av en egen algoritm, Picture-In-Garbage-Out (PIGO). Då pedagogik och grundläggande bildbehandlingsmetoder stod i fokus, snarare än effektivitet eller realistiska scenarion, applicerades detta på det digitala bildformatet BMP. De konsekvenser som detta resulterade i var främst nyfikenhet, förundran och en vilja att ta detta till en ytterligare nivå.

Den egna algoritmen PIGO tar inspiration från flera existerande steganografiska algoritmer men bidrar ändock med flertalet säkerhetsmässiga förbättringar. Möjligheten finns fortfarande att på många sätt förbättra algoritmen och dess tillämpningar än de som kort har presenterats häri. Den skulle således kunna vara del i en vidare analys där t.ex. möjligheten att tillämpa idén på destruktiva bildkomprimeringsalgoritmer kan undersökas. Följaktligen är området mycket intressant som mål för vidare studier eller andra angreppssätt och det är med vidgade ögon och ett ökat intresse vi härmed avslutar denna kandidatuppsats.

Litteraturförteckning

[1] *The King James Bible - Public Domain.* `http://patriot.net/~bmcgin/kjv12.txt`, 2002.

[2] *Molecular-Cancer.com, Figur.* `http://www.molecular-cancer.com/content/download/figures/1476-4598-5-67-5.BMP`, 2006.

[3] *Steganography, Wikipedia.* `http://en.wikipedia.org/wiki/Steganography`, 2009.

[4] *BMP file format, Wikipedia.* `http://en.wikipedia.org/wiki/BMP_file_format`, 2010.

[5] *JPEG, Wikipedia.* `http://en.wikipedia.org/wiki/jpeg`, 2010.

[6] *Portable Network Graphics, Wikipedia.* `http://en.wikipedia.org/wiki/Portable_Network_Graphics`, 2010.

[7] P. Bourke. Bmp image format. Technical report, The University of Western Australia, 1998.

[8] K. Hempstalk. Hiding behind corners: Using edges in images for better steganography. Technical report, Department of Computer Science, 2006.

[9] J. Lenti. Steganographic methods. Technical report, Department of Control Engineering and Information Technology, 2000.

[10] T. Morkel, J.H.P. Eloff, and M.S. Olivier. An overview of images steganography. Technical report, Department of Computer Science, 2006.

[11] N. Provos and P. Honeyman. Detecting steganographic content on the internet. Technical report, Center for Information Technology Integration, 2001.

[12] N. Provos and P. Honeyman. Hide and seek: An introduction to steganography. Technical report, Center for Information Technology Integration, 2003.

[13] F. Queirolo. Steganography in images. Technical report, 2001.