Ett system för lingvistisk steganografi genom synonymsubstitution

TOMAS ANDRÉASSON
och DANIEL HULTGREN

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

Ett system för lingvistisk steganografi genom synonymsubstitution

T O M A S A N D R É A S S O N
o c h D A N I E L H U L T G R E N

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

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/andreasson_tomas_OCH_hultgren_daniel_K10070.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

We have been studying various means of linguistic steganography: the science of hiding data in a seemingly innocent natural text. A possible use of this field is the transmission of secret messages using an insecure method of communication without arousing suspicion. The project has also involved creating, implementing and analyzing our own steganographic system.

Our system is based on the interchangeability of synonyms, a redundancy in natural language that can be used to store data. For each group of words with identical meaning, we can assign a binary string to each synonym, scan through a text and swap words with known synonyms for the ones with the binary string matching the next piece of data to be stored. A reverse procedure is used to read and output the binary data encoded in the text. Furthermore, we've also focused on corruption tolerance, that is, the ability to retain some or all encoded data if a portion of the text is removed or changed.

The tests of our system show a poor data density – on average one bit of raw data for every 37 words – caused by the restrictive synonym database used as well as an acceptable corruption tolerance for "natural" kinds of corruption, even though it quickly falls apart when a more chaotic method of destruction is applied.

Referat

Vi har undersökt diverse metoder för lingvistisk steganografi: läran om att dölja data i till synes oskyldig naturlig text. Ett möjligt användningsområde för detta är överföringen av hemliga meddelanden med en osäker kommunikationsmetod utan att väcka misstankar. Projektet har också involverat skapande, implementation och analys av ett eget steganografiskt system.

Vårt system baseras på synonymers utbytbarhet, en redundans i naturligt språk som kan utnyttjas för att lagra data. För varje grupp av ord med samma betydelse kan vi tilldela en binär sträng till varje synonym, gå igenom en text och byta ut orden med kända synonymer mot de vars binära sträng matchar nästa stycke data att lagra. En omvänd metod kan användas för att läsa och skriva ut den binära data som kodats i texten. Vidare har vi även fokuserat på korruptionstolerans, det vill säga förmågan att helt eller delvis återskapa den kodade datan om en del av texten raderats eller modifierats.

Testerna vi gjort på systemet visar en dålig datadensitet – i genomsnitt en bit rå data per 37 ord – orsakad av den restriktiva synonymdatabasen som använts samt en acceptabel korruptionstolerans för ”naturliga” sorters korruption, trots att den snabbt kollapsar vid mer kaotiska korruptionsmetoder.

Innehåll

Kapitel 1

Introduktion

Den här rapporten skrivs som en del av kandidatexamensarbetet vid Datateknik på KTH. Deltagande i arbetet är Daniel Hultgren och Tomas Andréasson.

Elektroniska verktyg används i allt större utsträckning för kommunikation. Internet har revolutionerat sättet vi kommunicerar på och tillåter fritt utbyte av idéer och åsikter. Det finns dock ett centralt problem: det är väldigt lätt att snappa upp meddelandet mellan avsändare och mottagare. Denna person (eller detta program) kan sedan enkelt läsa meddelandet, och om det inte faller honom i smaken kan han ändra det, eller vidta åtgärder mot personen som skickade det. Detta är en s.k. man-in-the-middle-attack. Det finns system som implementerats för att skydda mot detta, t.ex. SSL/TLS, men även dessa har sina svagheter[1]. Så länge krypering inte är standard kommer dess användning också väcka misstankar.

Steganografi handlar om att dölja data, vanligtvis i annan till synes oskyldig data. Lingvistisk steganografi är en underkategori som handlar om att dölja datan inuti naturlig text.

Den största delen av all data som skickas över Internet är ren text, så om man skickar ett meddelande som ser ut som ren text är risken låg att texten upptäcks av någon, och om det faktiska meddelandet ligger dolt i texten är risken ännu mindre att hemligheten uppdagas. Att det dessutom är mycket svårt för en dator att tolka naturlig text och därigenom hitta det hemliga meddelandet gör det ännu mer attraktivt. Därför är lingvistisk steganografi en lovande metod för att skicka dolda meddelanden.

Kapitel 2

Bakgrund

2.1 Kryptografi

Kryptografi är en mycket gammal konst. När den började användas kunde de flesta inte läsa, så chiffren var från början simpla.

Exempel på några enkla chiffer:

- **Caesarchiffer** går ut på att förskjuta bokstäverna ett antal steg.[2]
- **Substitutionschiffer** byter ut varje bokstav mot en annan, förbestämd, bokstav.[3]
- **Skytale-chiffret** går ut på att man rullar en remsa runt en pinne, varpå man skriver meddelandet på remsan. Sedan behöver man en pinne av samma tjocklek (nyckel) för att läsa meddelandet.[4]

Gemensamt för alla är att mottagaren måste ha rätt nyckel för att kunna dechiffrera meddelandet. Allt eftersom man utvecklade mer avancerade lösningsmetoder och hjälpmedel ökade sedan komplexiteten på chiffren, och sedan datorernas intåg har kryptografin blivit en mycket viktig forskningsgren som påverkar oss alla.

2.2 Steganografi

Steganografi är antagligen äldre än kryptografi, kanske rentav äldre än skrivkonsten i sig. Det går ut på att skriva meddelanden så att ingen utom avsändaren och mottagaren misstänker att det existerar. Om det man döljer meddelandet i är vanligt förekommande kan det lätt smita förbi oupptäckt i strömmen. Internet är en utmärkt plats att tillämpa steganografi på grund av dess stora flöde ”ointressant” information. Ett ”vanligt” exempel på steganografi är meddelanden dolda i bilder på Internet. Då bilder innehåller enkelt åtkomlig data som är relativt onödig kan denna ändras och meddelanden enkelt gömmas[5]. Steganografi kan även med fördel tillämpas tillsammans med kryptografi för dubbel säkerhet.

2.3 Lingvistisk steganografi

Att applicera steganografi på naturligt språk är en relativt ny idé, och det finns inte mycket forskning i ämnet. Då det cirkulerar oerhört mycket naturlig text på Internet är det dock en väldigt lovande teknik. Vissa problem finns dock, t.ex. att det är mycket svårt att lura en människa med sådana system på större skala, och det är väldigt bandbreddskrävande p.g.a. låg lagringskapacitet.

2.4 Problemdefinition

Vi planerar att först läsa in oss på befintliga system och idéer, utvärdera dem och baserat på den information vi samlat in – blandat med egna idéer – utforma och implementera ett eget system samt analysera dess funktionalitet.

Kapitel 3

Befintliga system

3.1 Introduktion

Lingvistisk steganografi är en relativt ny forskningsgren, och det finns bara ett fåtal implementerade system. Vi har läst in oss på ett antal, och presenterar ett urval av dem här.

3.2 Tyrannosaurus Lex

Winsteins system[6] baseras på substitution av synonymer. Varje ord har ett visst antal synonymer och genom att numrera, översätta dem till sin binära motsvarighet och välja den som matchar data kan meddelandet lagras. Exempel på hur detta kan se ut beskrivs i 4.3.1. Den substitutionen ger dock i sin naiva form ett ganska dåligt resultat då den lagrar all data från vänster till höger och sedan slutar lagra data när den är färdig. Resultatet blir att texten ser mer eller mindre konstig ut i början för att bli helt normal mot slutet. Han föreslår istället ett system där han delar in det som ska gömmas och de möjliga synonymerna i block. Genom att göra detta kan man se hur mycket kapacitet man har i texten och då genom att fylla ut blocken med den överflödiga kapaciteten fördela hemligheten över hela texten, vilket ger en betydligt mer naturlig text. Winsteins system ger också möjligheten att välja vilka substitutioner som ska göras, vilket kan resultera i bättre texter då en människa oftast väljer bättre ord än en dator.

3.3 Mimicry

Wayners system[7] gömmer inte data genom att modifiera texter som ovan, utan genom att generera hela meddelandet efter ett antal fördefinierade regler. Systemet kan därmed beskrivas som en stor meningsmall. Med hjälp av en kontextfri grammatik som definierar hur ord, fraser och meningar kan ”klippas och klistras” ihop, ofta med flera alternativ att välja mellan, kan då en större text genereras. Genom

att sedan matcha texten mot mallen kan man se vilket val som gjorts vid varje ”vägskäl” i grammatiken, och utifrån detta utvinna det hemliga meddelandet.

Det som kanske är hans kanske mest kända exempel på detta system är den s.k. baseballgramatiken[8] som simulerar en sportkommentators beskrivning av en baseballmatch. De karakteristiskt korthuggna och smått kaotiska meningarna från sport-
kommentatorer gör det lättare att förbise de ”lätt” osammanhängande texter som systemet genererar. Ett utdrag från resultatet då ”KTH” dolts:

> Well Bob, Welcome to yet another game between the Whappersand the Blogs here in scenic downtown Blovonia . I think it is fair to say that there is plenty of BlogFever brewing in the stands as the hometown comes out to root for its favorites. The Umpire throws out the ball . Start of another inning . No damage yet, Mike . The crowd is nervous. Sal Sauvignon swings the bat to get ready and enters the batter's box . Here we go . Whoa, he's brushing him back . No strike this time . He's uncorking a knuckler . High and outside . Definitely a ball . He's winding up . What a fastball with wings . drives it into the stands for a HomeRun! Baseball and Apple Pie . Here we go. Albert Ancien-Regime adjusts the cup and enters the batter's box . Yeah. [...]

Exempel 1.

Genom att studera vägvalen i texten och de möjliga valen i grammatiken kan meddelandet avkodas. Den första delen (fram till ordet ”favorites”) är en av två möjliga introduktioner. Meningen ”The Umpire throws out the ball” är sen ett av fyra alternativ för att starta en runda som programmet kan välja, följt av en av fem fortsättningar som i detta fall blev ”Start of another inning”. Allt eftersom texten fortsätter förgrenas trädet av möjliga val allt mer och i löven på det i slutändan enorma trädet ligger olika meddelanden. Vägen genom trädet som denna text tar leder just till lövet ”KTH”.

Kapitel 4

Vårt system - WaxTablet

4.1 Våra prioriteringar

I prioritetsordning:

4.1.1 Mänsklig och automatisk analys

Den huvudsaklinga poängen med ett steganografiskt system är att en iaktagare inte ska se att det finns en kod i mediet (i det här fallet text). Detta är vår huvudsakliga prioritet, WaxTablet ska klara både mänsklig och maskinell analys. Dock har vi ingen möjlighet att testa den sistnämnda då vi inte hittat något öppet system för detta, och att koda ett eget fungerar inte då vi har facit i hand och därmed inte kan skapa ett analysprogram på ett opartiskt sätt.

4.1.2 Motståndkraft mot ändringar

Om någon eller något snappar upp texten innan den nått sitt slutgiltiga mål är ju tanken att denne inte ska se att det finns något dolt i den. Det är dock ändå möjligt att denne ändrar lite i texten för att eventuell hemlig kommunikation ska gå förlorad, eller kanske rentav tar bort bitar av texten. WaxTablet ska i största möjliga mån klara detta, och ändå leverera ett läsbart meddelande.

4.1.3 Informationsdensitet

Ju mer information man packar in i texten desto större är risken att någon upptäcker att den finns där. Informationsdensitet går inte heller så bra ihop med motståndskraft mot ändringar då man behöver redundant data för att klara det. Vi prioriterar detta lägre än de två första.

4.1.4 Kryptering

Meddelandet i sig är dolt, men utöver detta kan man kryptera det. Detta är inget vi prioriterar då det inte är helt relaterat till den steganografiska delen. WaxTablet

har dock stöd för enkel XOR-kryptering (en krypteringsform där datan XOR:as med ett värde som baseras på lösenordet), och tillåter att man själv lägger in en egen krypteringsalgoritm om man så skulle önska.

4.1.5 Översättning

Att kunna återskapa data efter att den modifierade texten översatts är eftertraktansvärt och något vi gärna skulle stödja, men det sätter stora begränsningar på resten av systemet. Detta är något som är viktigt för system vars huvudsakliga uppgift inte är att lagra utan att märka data, som vattenmärkningssystem. WaxTablet är inte i huvudsak till för det, så vi prioriterar inte detta.

4.2 Analys av systemidéer

Baserat på studier av befintliga system samt egna idéer fick vi tidigare fram en lista över diverse tillvägagångssätt för steganografiska system. Somliga lämpade sig bättre än andra med avseende på våra mål, vår kompetens och tidsåtgången för att implementera idén i fråga.

4.2.1 Mallbaserat system

Mallbaserade system som Chapmans NICETEXT[9] och Wayners Mimicry[7] bygger enligt tidigare på att använda meningsmallar för att koda data. Detta gör att de automatiskt kan generera texter utan att ha en originaltext, något som gör att de kan tillämpas på större skala. Dessutom har de hög datadensitet (d.v.s. den datamängd som kan lagras per textmängd). Den stora nackdelen är dock att texterna i regel ser väldigt kaotiska ut och därmed inte skulle passera en mänsklig kontroll. Dock kan en dator relativt enkelt luras då texten är grammatiskt och statistiskt korrekt (då datoranalyser i regel bygger på grammatik och statistik).

Det största kravet på detta system är en stor databas över meningsmallar (för att inte ge alla meningar samma, förmodligen korthuggna, struktur), ord samt data om vilka ord som passar in i vilka sammanhang. Detta kan vara svårt att få till på ett bra sätt och även då är det oerhört stor risk att en människa reagerar på resultatet. En väldefinierad kontext som tillåter korthuggna meningar och osammanhängande texter kan dock minska dessa problem, något som Wayner visade med sitt baseballinspirerade system[8]. Detta visas i Exempel 1.

Mycket arbete för att generera stora databaser för resultat som lätt fångas av människor gjorde att vi valde att inte använda detta system.

4.2.2 Synonymbaserat system

Definition av synonym

En synonym definieras antingen som ett ord som man i *alla tänkbara fall* kan byta mot ett annat utan att förändra textens mening(som vanligtvis tillskrivs Leibniz)

eller ett ord som man man bara kan byta mot ett annat i *ett fall* och behålla meningen[10]. Använder man den första definitionen är synonymer sällsynta men substitutionen perfekt, vilket i ett steganografiskt sammanhang betyder att man kan få perfekt substitution men väldigt låg lagringskapacitet. Den andra definitionen ger betydligt fler synonymer, men minskar kvaliteten på synonymerna.

Analys

Synonymbaserade system kräver en originaltext att modifiera, men detta gör också att de i slutändan får vettig semantik och en tydlig röd tråd. Detta gör det betydligt svårare för både människor och datorer att misstänka texten, något som också gör detta till en potent teknik för vattenmärkning av text. En annan nackdel är att de, då bra synonymer är sällsynta, ger lägre datadensitet än mallbaserade system. Winsteins Tyrannosaurus Lex är ett exempel på ett synonymbaserat system[6].

Ett krav på detta system är en stor databas över synonymer, helst synonymer med exakt samma betydelse. Dessa är dock extremt sällsynta, så man får välja hur bra synonymer man vill ha. Bättre synonymer ger lägre datadensitet, men ökar sannolikheten att resultatet skulle lura en människa. Ett problem är dock att alla synonymer inte är lika sannolika att finna i en text, något som i längre texter kan skapa en svaghet mot statistik analys.

Slutsatsen är att detta verkar vara ett väldigt lovande system då det är svårt för mänskliga såväl som automatiska analyser att misstänka texten.

En utbyggnad av systemet kan också göras som bygger på ”meningssynonymer”, d.v.s. där vi även gör synonymer på meningsnivå genom att ha en databas över hur meningar kan modifieras och genom detta lagra data. Teoretiskt tyckte vi att detta skulle vara en bra idé, men en databas stor nog att rättfärdiga dess skapande och implementation bedömde vi vara för tidskrävande för att hinna med inom de tidsramar vi fått.

4.2.3 Ordklassbaserat system

Att koda datagram baserat på fördelningen av ordklasser gör meningsstrukturen oerhört begränsad och att automatiskt omforma en text är onödigt komplicerat. En fördel med detta system är dock att det skulle kunna göra det möjligt att utvinna viss data efter översättning, förutsatt att språken är snarlika nog att nästan ha ett ett-till-ett-förhållande mellan de ursprungliga och översatta orden och därmed bibehålla ordklassfördelningen. På så sätt skulle förmodligen svenska till norska eller italienska till spanska fungera bra.

Förmågan att översätta väger dock inte tyngre än svårigheterna, så vi valde att inte använda ett sådant system.

4.2.4 Övriga idéer

Förutom de ovan nämnda systemen hade vi ett antal andra idéer, här är ett urval:

- Ett system där data lagras i valet av ordlängder. Det är dock svårt att generera och väldigt komplicerat att omforma en existerande text. Dessutom blir resultattexten i regel alltid oerhört krystad och lurar inte människor.
- Att koda en bit per ord beroende på om första bokstaven är en vokal eller konsonant. Betydligt lättare än systemet ovan men fortfarande svårare än synonymer, och lite väl uppenbart.

4.3 Beskrivning av vårt system

Som vi tidigare nämnt är en av våra prioriteringar robusthet i form av möjligheten att helt eller delvis återskapa data om texten på något sätt förstörts, till exempel genom citering av en del eller på annat sätt modifierats av en tredje part. För att uppnå detta mål har vi skapat ett system byggt i ett antal ”lager” lagda ovanpå varandra. Personen som skapar meddelandet kan välja vilka lager som ska användas så länge alla underliggande lager används; valet handlar om en avvägning mellan robusthet och storlek.

Vid kodning använder man lagren i stigande ordning, från det lägsta till det högsta önskade, där varje lagers utdata blir indata för nästa. Omvänd ordning gäller för avkodning.

4.3.1 Lager 1 – Kodning av binärdata i synonymer

Det steganografisystem som vi ansåg vara bäst lämpat för detta arbete blev ett där datagram lagras i texter genom att utnyttja synonymers utbytbarhet.

Som beskrivet tidigare i rapporten fungerar systemet genom att ha en stor databas över synonymer, ge alla ord i varje synonymgrupp ett unikt värde, söka igenom texten efter ord som har synonymer och ändra dessa till den synonym som har samma värde som de aktuella bitarna i indatan att lagra.

I pseudokod ser kodning av data ut såhär:

Indata:
    Binärsträng B innehållandes datan att dölja i texten
    Lista W innehållandes måltexten, uppdelad till lista med ett ord per element
    Databas D över synonymer
Medan vi har bitar kvar i B:
    Om det inte finns några element kvar i W:
        Returnera felmeddelande: Datan får inte plats i måltexten
    Om nästa ord O i W har en synonym:
        Hitta den synonym S till O vars värde är det
        längsta som matchar nästföljande bitar i B
        Ändra O i W till S
Returnera W

Pseudokod 1.

På liknande sätt kan man plocka fram den binära datan ur en steganografisk text.

Vår synonymdatabas

För att systemet ska fungera behöver man såklart en databas över synonymer, och både avsändare och mottagare måste ha tillgång till denna. Vi har valt att använda den databas som Winstein utvecklade och använde för sin Tyrannosaurus Lex. Den är baserad på WordNet, men han har sedan delat upp synonymerna i synonymgrupper där alla ord är synonymer till alla andra, och ignorerat synonymer med mellanslag. Om ordet har flera betydelser har han bara sparat de ord som är synonymer till båda betydelserna. Mer utförlig definition finns att läsa i hans rapport[6]. Resultatet blir att ungefär 70 % av alla WordNets synonymer tappas, men kvaliteten på synonymerna blir i gengäld hög. Man skulle också kunna tänka sig att tillåta synonymer som beror på kontext för att få bättre datadensitet på kostnad av synonymkvaliteten, något som vi dock valde att inte göra.

Exempel på datalagring

Algoritmen kommer enligt tidigare fram till den synonym som kan lagra flest bitar av den data som kan lagras. Här är ett exempel:
Data som ska lagras: 001...
Aktuellt ord: ”nothing”
Synonymer: {aught, nada, nix, nothing, zilch}

Dessa synonymer är i bokstavsordning, och samlingen är densamma för alla synonymer i den (enligt tidigare är den reflexiv). Då vi har fyra synonymer är man garanterad att kunna lagra två bitar, så längden på värdena blir 2 bitar, förutom den sista som blir 100 (om datan råkar matcha detta kan man lagra tre bitar). ”aught” kommer alltså representera värdet 00, upp till ”zilch” som representerar värdet 100. Som vi ser på indatan är det bästa vi kan lagra 00, och algoritmen kommer därmed välja ”aught” och byta ut ”nothing” mot aught i texten.

Originaltexten är följande: ”The campaign brought honours and promotion to many, but for me it had nothing but misfortune and disaster”.

Resultatet blir: ”The campaign brought honours and promotion to many, but for me it had aught but misfortune and disaster”.

Ordvalet kanske verkar underligt och skulle vara det i en mer modern text, men i kontexten (detta är en bok från 1887) känns det mer naturligt.

4.3.2 Lager 2 – Lagring av data i block

Vårt mål om att få ett robust system måste klara av att bitar försvinner eller tillkommer, vilket ofta är fallet om någon redigerar texten på ett eller annat sätt. Om så är fallet finns det en stor sannolikhet att bitarna förskjuts och därmed feltolkas.

För att kompensera för detta delar vi in datan i ett antal ”block”, en datastruktur innehållandes ett fixt antal bitar rå data samt en checksum och ett ID-nummer. Värt att notera är att ID-numret endast läggs in om man även använder sig av lager 3 då det inte behövs för lager 2.

Efter att den råa binärdatan extraherats från lager 1 stegar man igenom binärsträngen bit för bit och kontrollerar om det är ett giltigt block. Stämmer checksumman lägger man till blocket i en lista, hoppar fram till biten efter blocket och upprepar. På så sätt kastar man bort ogiltiga block som på ett eller annat sätt förstörts.

Blocken har följande struktur:

C bitar: Checksum

L bitar: ID (om lager 3 används)

D bitar: Rå data

...där C, L och D är parametrar som man kan finjustera efter personliga önskemål. C låter en välja mellan större utdata och större risk att av misstag acceptera ett trasigt block. D låter en välja mellan större utdata och större förlust av data ifall ett block går sönder. L beror på D och mängden data att gömma. Då det sistnämnda kan vara svårt att bestämma i förväg kan man ha något standardvärde beroende på hur långa meddelanden man tenderar att skicka.

Exempel på blockläsning

Här kommer ett exempel på hur vi använder blockstrukturen för att förkasta skadad data. I exemplet antar vi att C = 3, L = 5 och D = 6.
Data som läses in: 1010100000010100...
Tolkning som block: 101 01000 000101 (Checksum ID Data)

Av detta beräknar vi checksumman och får 011, vilket inte matchar den checksumma som finns i blocket. Blocket är ogiltigt och vi stegar fram ett steg i indatan och upprepar:
Data som läses in: 010100000010100...
Tolkning som block: 010 10000 001010 (Checksum ID Data)

Även här blir checksumman (101) fel, och vi stegar vidare till nästa bit.
Data som läses in: 10100000010100...
Tolkning som block: 101 00000 010100 (Checksum ID Data)

Här får vi checksumman 101 som matchar blockets checksumma. Blocket klassas därför som korrekt och sparas. Inläsningen fortsätter efter blockets slut.

4.3.3 Lager 3 – Redundans av block

Vi har tidigare gjort det möjligt att lagra data (lager 1) och fortsätta läsa data även om man tappat delar av texten (lager 2). Det kan även vara önskvärt att återskapa förlorad data, och för detta finns lager 3. Vi har valt att implementera det på ett mycket enkelt sätt: vi skriver helt enkelt datan tre gånger. Sedan parar vi ihop all data som har samma ID och väljer den data som det finns flest av.

Detta steg skulle kunna effektiviseras avsevärt, och är man intresserad kan man byta steg 3 mot någon annan, effektivare algoritm. Detta skulle kunna resultera i bättre möjligheter för återskapning och större lagringskapacitet.

4.3.4 Kryptering

Om en tredje part misstänker att datagram finns lagrade i texten och känner till steganografisystemet är spelet slut eftersom texten mer eller mindre är lagrad i klartext när alla lager skalats bort. Även om texten är dold kan man anse det vara onödigt att inte kryptera texten ”bara ifall att”. Kryptering kan göras utan att extra utrymme i texten krävs, så det finns inga nackdelar (bortsett från ”besväret” att behöva knappa in ett lösenord för att koda/avkoda meddelandet). Därför har vi lagt till ett frivilligt krypteringssystem.

Då kryptering inte är det huvudsakliga syftet med detta arbete använder vi bara ett enkelt (och lättknäckt) XOR-krypto. Då vi inte känner till något avancerat kryptografisystem som klarar dataförlust använder vi den naiva metoden att ta MD5-hashen av lösenordet, plocka ut de nedersta X bitarna (där X är längden på den faktiska datan i blocken) och XOR:a datan i blocken med detta. Detta görs innan checksumman beräknas, så att den därmed reflekterar den krypterade datan.

4.3.5 Teckenkodning

Då man kommer ha relativt begränsat lagringsutrymme i texterna föreslår vi ett enkelt teckenkodningssystem på sex bitar per tecken. Systemet stödjer stora och små bokstäver (A-Z), siffror, mellanslag och punkt:

Tecken 0x00 – 0x09: Siffror (0 - 9)
Tecken 0x0A – 0x23: Versaler (A- Z)
Tecken 0x24 – 0x3D: Gemener (a - z)
Tecken 0x3E: Mellanslag
Tecken 0x3F: Punkt

Ett mer omfattande system som ASCII eller till och med UTF-8 skulle lätt kunna användas i stället, men skulle så klart kräva mer utrymme och därmed längre texter.

Om ytterligare utrymme måste sparas följer här ett förslag på en teckenkodning på fem bitar:

Tecken 0x00 – 0x19: Versaler (A - Z)
Tecken 0x1A: Mellanslag
Tecken 0x1B: Punkt
Tecken 0x1C – 0x1F: (Ospecificerade tecken)

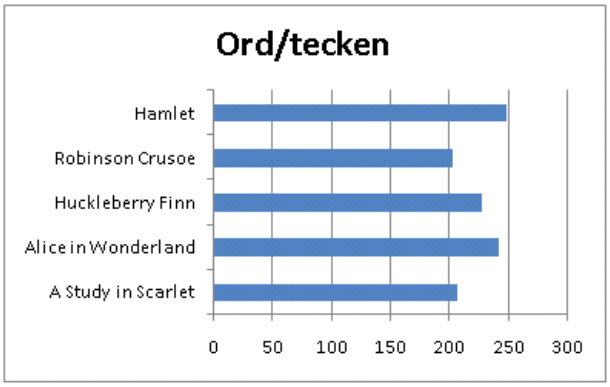
Kapitel 5

Resultat

5.1 Densitetstest

I vår slutgiltiga version använder vi databasen från Winsteins Tyrannosaurus Lex[6]. Den är till skillnad från WordNet reflexiv, ett krav för entydig avkodning. Antalet synonymer är dock betydligt reducerat då alla synonymer måste överensstämma helt. Resultatet av databasbytet blev därmed en väsentligt försämrad (lägre) datadensitet. Alternativet är att göra en egen databas baserad på WordNet som har färre restriktioner än Winsteins, eller att tillåta tvetydig avkodning. En mindre restriktiv databas skulle dock ha nackdelen att synonymerna inte passar lika bra, vilket gör det lättare för en människa att misstänka texten.

Datadensiteten beror på hur många, och vilka, av databasens synonymer som används. Vi testade därför densiteten med ett antal gratis tillgängliga böcker.



Figur 5.1. Datadensitet i olika böcker.

Av testerna att döma kan vi räkna med att varje bokstav (lagrad rå utan någon redundans) tar mellan 200-250 ord att lagra, eller 33-42 ord per bit. Detta antal kan enligt tidigare minskas om man använder en mer ”aggressiv” databas.

5.2 Rekonstruktionstest

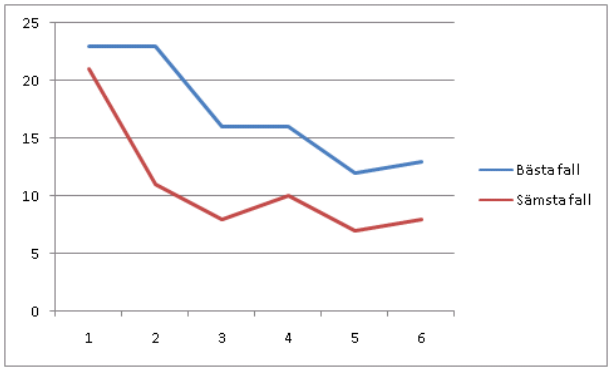
För att testa lager 3 använde vi lite olika anfallsvinklar. Vi testade att ta bort slumpmässiga ord respektive kapitel och citera enbart enstaka kapitel. Alla tester gjordes på ”A Study in Scarlet” av A. C. Doyle. Testerna utfördes genom att koda meddelandet ”Mary had a little lamb.”, ett meddelande som nästan helt fyllde boken (redundant data inräknad). Detta för att WaxTablet automatiskt fyller texten med extra redundans i mån av plats, något som skulle förvränga resultatet.

5.2.1 Borttagning av slumpmässiga ord

Vi gjorde ett testprogram som slumpmässigt tog bort en viss procent av alla ord i boken som meddelandet kodats i, detta för att testa motståndskraft mot korruption. Resultatet var under förväntan, men vi förstår varför det blev så. Vi körde varje procent tio gånger och valde subjektivt de bästa respektive sämsta resultaten:

%	Bästa fall	Sämsta fall
1	Mary had a little lamb.8C1mYn4	Mary hd y little lamb.H2e6m11mKXG T4n
2	Mary had a little lamb.XeG6Pm4OYX3G X44	Mar Ad 8 kHylamc.0Re611qX3mKCYX T2W0
3	Mary had Mi lamb.0Rm17mXK3zX4	Kry hayk8ylamyNRAe611mmOK6aXSx34G b594
4	Mary Au alitk lamb.2UR6mWfF4mKYX PXZ2O	SPru jaW tEe la.U19R.OqmyYXsLtPX4W4
5	Mh Ad a l. lambWO1Re6m13m4mKY137 TSX4D4	Lrvyd yymMamA.1yR6m11O4F4mmGzTd4Q409
6	Mry hA8 lil laLb.TokTe62h1F1Qm7mOO377GJP2	Crv h8 ll lnKHA2R3AeC1fn4mYs3GP2X9250Wn

Figur 5.2. Subjektivt bästa resp. sämsta resultaten av slumpmässig korrumpering.



Figur 5.3. Approximativt (delvis subjektivt) antal korrekta tecken av 23.

Som man kan se försämras resultatet snabbt, och redan vid 2% var det i värsta fall oläsligt. Notera dock att WaxTablet inte har något sätt att markera att texten är slut, så de allra flesta avkodade meddelanden slutar med skräpdata, något man till stor del slipper med längre checksum. Dessa tecken ska därmed inte räknas med när man jämför meddelandenas kvalitet.

Resultatet blir dåligt eftersom sannolikheten är relativt stor att det slumpmässigt valda ordet är ett kodord, och när ett kodord tappas förstörs hela dess block (i

dagsläget bestående av 16 bitar), och därmed går hela bokstaven förlorad. Redundansen kan rädda till viss del, men snart har även den redundanta datan förstörts. Att resultatet på det här testet skulle bli dåligt anade vi utifrån liknande system, men att det skulle bli så här dåligt var dock oväntat.

5.2.2 Borttagning av slumpmässiga kapitel

För att testa mer linjär borttagning av text plockade vi bort kapitel pseudoslumpmässigt. Boken har två delar med vardera 7 kapitel, som benämns 1.1-1.7 resp. 2.1-2.7.

Borttagna kapitel	Resultat
1.1	Mary had a little lamb.I
1.4, 1.5, 1.7, 2.5, 2.7	Mary had a little lamb.
1.2, 1.3, 1.4, 1.5, 1.7, 2.2, 2.5, 2.7	Mary hZd a l lama.Rq4K T2W
1.3, 1.5, 1.6, 2.2, 2.4	Mary had a little lamb.4R3K TW
1.1, 1.3, 1.5, 1.6, 2.2, 2.3, 2.4	Mard a little lamb.4e3IK T

Figur 5.4. Resultat vid pseudoslumpmässig borttagning av hela kapitel.

Som vi kan se i figur 5.4 klarar texten den här sortens korruption betydligt bättre än den helt slumpmässiga. Detta p.g.a. att en stor mängd data kan tas bort utan att mer än en kopia av datan förstörs. Faktum är att med denna metod kan man ta bort de första två tredjedelarna av filen och fortfarande få i stort sett rätt resultat.

5.2.3 Citering av enstaka kapitel

Vi testade också motsatsen, d.v.s. att plocka ut enstaka kapitel och se vad vi fick ut från dem. Detta eftersom det inte är alltför sällsynt att man citerar delar av en text. Det intressanta är att vi ibland fick ut ganska vettig data från det, om än i väldigt små doser.

Kapitel	Resultat
1.2	lmG4
1.4	amb
1.6	ry h
2.7	a

Figur 5.5. Resultat vid pseudoslumpmässig citering av enstaka kapitel

I figur 5.5 lagrar 1.4 och 1.6 uppenbarligen ”Mary had a little lamb.” resp. ”Mary had a little lamb.”. 1.2 blev inte lika bra, då bitarna blivit förskjutna och checksumman inte räckte till för att förhindra feltolkning. 2.7 innehåller antagligen början av meddelandet igen, då den hunnit börja om precis på slutet (lager 3 lagrar extra redundant data i mån av plats).

5.3 Slutsats

Som nämnts tidigare i 4.2.2 har WaxTablet en svaghet i och med dess fördelning av synonymer. Sättet som synonymerna väljs på gör att de som kommer tidigt i bokstavsordning har aningen större sannolikhet att uppstå, även om synonymen i sig sällan förekommer i verkligheten. En dator som kollar ordvalsstatistik kan därmed reagera och flagga texten som underlig. Bergmair nämner en lösning i sin avhandling[11] där han föreslår ett system som avgör hur många bitar varje ord representerar – och därmed hur vanliga de blir – baserad på statistisk data över det engelska språket.

Som vi tidigare konstaterat i kapitel 5.2.1 tappar WaxTablet snabbt data vid borttagning/korrumpering av slumpmässigt valda ord. Detta då ett helt block (i dagsläget 16 bitar) tappas så fort en bit går förlorad eller ändras. Detta skulle kunna effektiviseras genom annorlunda teckenkodning där mindre block används, eller med en helt annan lagringsteknik.

En annan lagringsteknik (kanske någon form av komprimering såsom Huffmankodning[12]) skulle också kunna förbättra lagringskapaciteten vilken var riktigt dålig i våra tester, på gränsen till oanvändbar i praktiska sammanhang annat än vattenmärkning. Detta skulle även kunna lösas genom att använda en mer ”aggressiv” databas (en som tillåter friare men mindre korrekta synonymer). Man skulle också kunna använda en ickereflexiv databas, något som skulle ge betydligt högre kapacitet på bekostnad av att datan blir tvetydig och man alltså kommer behöva gissa vad originalmeddelandet var, något som förespråkas av Bergmair[11]. Detta kan göras t.ex. genom att jämföra meddelandetexten mot ordlistor eller presentera olika alternativ för användaren.

Redundansen (lager 3) är i dagsläget naiv och har därmed mycket utrymme för förbättringar. Det finns system för förlusttolerant data, och dessa kan anpassas till WaxTablet. Denna avhandling fokuserar på steganografi, men för ett system som ska användas i verkligheten vill man antagligen använda ett mer avancerat kryptografisystem. Systemet i fråga måste dock tolerera förlust av data, vilket begränsar valmöjligheterna avsevärt. En avsevärd förbättring vore även att lagra storleken på checksum och id direkt i den kodade texten, så slipper man komma överens om dessa i förväg. Går de förlorade kommer dock hela texten gå förlorad, eller i bästa fall bli tvetydig.

För att minska dataförlusten vid längre meddelanden kan man slumpa ut bokstävernas placering i texten (eftersom vi ger varje block ett id-nummer kan det sorteras tillbaka till sin ursprungliga position). Vid slumpmässig borttagning av ord bör detta inte göra någon skillnad, men om man tar bort större delar kommer detta sprida ut felen över hela texten. Med andra ord förlorar vi inte flera ord på rad utan någon enstaka bokstav per ord i hela texten, vilket bör göra texten mer lättförståelig.

Slutligen kan vi säga att WaxTablet är fullt användbart idag, men det skulle kunna göras bättre, framför allt genom att öka datadensiteten.

Litteraturförteckning

[1] P. Burkholder. Ssl man-in-the-middle attacks. `http://www.sans.org/reading_room/whitepapers/threats/480.php`, 2002-02-01. Nerladdad 2010-03-17.

[2] F. Richman. Shift ciphers. `http://math.fau.edu/Richman/caesar.htm`, 2005-03-15. Nerladdad 2010-04-14.

[3] Z. Stankova-Frenkel. Simple substitution ciphers. `http://mathcircle.berkeley.edu/BMC3/crypto/node2.html`, 2000-12-17. Nerladdad 2010-04-14.

[4] A. Howard. Skytales. `http://banach.millersville.edu/~bob/math478/History/Skytales.html`, 2004-09-09. Nerladdad 2010-04-14.

[5] N. Memon M. Kharrazi, H. T. Sencar. Image steganography: Concepts and practice. *Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*, 2004-04-22. Nerladdad 2010-03-17.

[6] K. Winstein. Lexical steganography through adaptive modulation of the word choice hash, Januari 1998. Opublicerad.

[7] P. Wayner. *Disappearing Cryptography – Information Hiding: Steganography & Watermarking*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, second edition, 2002.

[8] P. Wayner. Mimicry applet. http://www.wayner.org/texts/mimic/, 1997-08-20. Nerladdad 2010-03-10.

[9] M. T. Chapman. Hiding the hidden: A software system for concealing ciphertext as innocuous text. Master's thesis, University of Wisconsin-Milwaukee, Maj 1997. Nerladdad 2010-03-17.

[10] C. Fellbaum D. Gross K. Miller G. A. Miller, R. Beckwith. Introduction to wordnet: An on-line lexical database. *Int J Lexicography*, 3(4):235–244, 1990.

[11] R. Bergmair. Towards linguistic steganography: A systematic investigation of approaches, systems, and issues. Examensuppsats, April 2004. Nerladdad 2010-01-03.

[12] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E*, 40(9):1098–1102, 2007-01-08.