StegChat: A Synonym-Substitution Based Algorithm for Text Steganography

Joseph Gardiner
Supervisor: Dr. Shishir Nagaraja

Submitted in conformity with the requirements
for the degree of MSc Computer Security
School of Computer Science
University of Birmingham

Abstract

StegChat: A Synonym-Substitution Based Algorithm for Text Steganography

Joseph Gardiner

Steganography, the art of information hiding, has been around for thousands of years, with the earliest examples coming from as early as 450 B.C. Modern steganography can be applied to text, images audio and video. Text, however, has received less attention in recent years, primarily due to the lower capacity to hide information than the other mediums. This should not be the case, as text steganography has many benefits over the other mediums which make it ideal for effective steganography. One advantage of text steganography over images and audio is that while they are both susceptible to compression due to their use of redundant data, this is not an issue with text steganography as even though text contains redundancy, it can not be removed or compressed. Text is also still one of the major forms of communication in the world, both in digital and printed form, and there are not many people who do not have access to text.

In this project I propose a lightweight and robust algorithm for text steganography using the idea of synonym substitution. The algorithm will be demonstrated using a prototype chat-based application, StegChat, and evaluated for its resistance to both automatic and human analysis.

Acknowledgments

I would first like to extend a special thanks to my supervisor, Dr. Shishir Nagaraja, for his support and guidance throughout this project. I would also like to thank my family and friends for all of their patience and support during this project.

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Birmingham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Signed .....

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Steganography, the art of information hiding, has been around for thousands of years. The first recorded use was in 450 B.C , where Demaratus sent messages hidden beneath the wax on writing tablets.

Steganography is concerned with hiding the existence of data by encapsulating it within some covertext. The goal is to make the hidden data undetectable, by man or machine. If the existence of hidden data can be proven, then the steganography can be considered to have failed, even if the hidden data itself is not recovered. This is in contrast to cryptography, where the goal is to prevent the data from being revealed, with no attempt to hide the existence of the data (in fact cryptographic data is often easily identified). Cryptography can be used in conjunction with steganography to provide an added layer of security in case the steganography is broken.

Modern steganography can be applied to a number of mediums. Text, images, audio and even video are all commonly used as carriers for secret messages. Text steganography, however, has received less attention in recent years, mainly due to lower capacity to hide information that is associated with it. There are many reasons why text steganography deserves continued research and development. For one, text is still the primary form of communication in many areas of the world where computers and the internet are not widespread. Text is also universally applicable, nobody has images or audio who does not have text.

Perhaps the unique feature of text steganography over steganography which makes use of images or audio is that image and audio steganography uses redundancy in the data. This redundancy can be easily removed during compression, which would destroy any hidden data. On the other hand, while text steganography uses the English language, which still contains redundancy, such as synonyms, this redundancy cannot be removed since the rules of the language are fixed.

In this project I propose an lightweight, robust algorithm for text steganography which uses the idea of synonym substitution i.e. hiding bits by replacing words with words of the same (or similar) meaning. This algorithm will be tested using a simple chat-based application,

StegChat, and will be evaluated against both statistical and human analysis.
This project is a research-based project, with an emphasis on the algorithm and design rather than the implementation of the program.

1.1 Motivation

Steganography, as an area, has been the focus of less research in recent years with more time spent on improving cryptography. As will be discussed in the next chapter, this should not be the case as steganography can be a very powerful tool, especially if used in conjunction with cryptography. Text steganography is now the least popular form of steganography among researchers and commercial products. There are a number of possible reasons for this, including a much lower capacity to hold data when compared to mediums such as images or audio, but there are some reasons why it should gain more attention.
Firstly, in many parts of the word text is still the primary form of communication. This can be either through the printed word, simple messaging service (SMS) on mobile phones or email. Computers and access to the internet may also be very limited, so a steganographic technique must be suitable to be performed by hand if required (trying to insert data into the bits of an image without a computer is near enough impossible).
Text is a medium that is universally applicable. Nobody has access to digital images or audio who does not also have some access to some form of text, whether it be the printed word or digital text in the form of emails or SMS messages. Finally, text steganography, especially of the semantic form, is much more resistant to external forces than other mediums. Compression algorithms can have devastating effects of data hidden in images or audio, but you cannot compress text without losing the meaning. There is also a larger amount of variation in text compared to the other forms, making it more resistant to Steganalysis. For example, editing every kth bit in an image can be easily found by comparing it with the surrounding bits, if it is dramatically different then it will be noticed. With text, however, variations in language, even within different regions of a country with one unified language, can help to disguise hidden data, for example through synonym substitution.

1.2 Aims

The primary aim of this project is to produce a lightweight and robust steganography technique, and then to produce a demonstration application to test the algorithm in a real-life situation. The algorithm should also be simple enough that an end user can understand it without any specific knowledge. To elaborate on these points

Lightweight

The algorithm should be lightweight in two senses. The algorithm should be simple enough that it can be performed by a human using a printed dictionary. This is primarily so that the algorithm can be performed where there is no access to a computer or the internet. It should also be lightweight enough that it can be run in almost real time for the chat-based application, and can be used on a device with little computing power.

Robust

The algorithm needs to be robust against two forms of steganalysis, automatic and human. The output of the algorithm should pass any statistical steganalysis method, including frequency analysis. The text that is outputted should also be readable by humans, and should not arise any suspicion due to incorrect word replacements or a loss of meaning.

The primary aim of the project is for more robustness against steganalysis rather than a higher bitrate. The desired bitrate is around 1 bit for every 10 words, which is in line with the approaches described in section 2.3.

1.3 The Report

This report will consist of four further parts. First, there will be an explanation of steganography with a focus on text steganography, and a literature review of current research into synonym-based steganography (and similar methods). Next will be a full explanation of the design of the algorithm, and a brief design of the StegChat test application. Thirdly, there will be an section on the implementation of the algorithm and test program, including the problems implementing the algorithm and a description of the structure and operation of the program. This will be followed by an evaluation of the algorithm, and finally a conclusion.

1.4 Glossary

Synset - A set of synonyms.
Obfuscation - The act of hiding data within a covertext.
Deobfuscation - The act of extracting data hidden in a covertext.

Chapter 2

Background Information and Research

2.1 Steganography

2.1.1 History of Steganography

The term "Steganography" comes from the word "Steganographia", the title of a series of three books by the German abbot Johannes Trithemius [36]. While the books appeared to be about magic (in particular using the spirits to communicate over long distances) the books actually contained hidden information on cryptography and steganography.
One of the earliest examples of steganography can be found in around 440 B.C., where Herodotus (c.486-425 B.C.) wrote in his *Histories* about how Histiæus would shave the head of one of his most trusted slaves and then onto it would tattoo a secret message. Once the hair had regrown, the message was hidden and the slave could be sent to deliver the message. [27]
Another more modern example is from the Second World War. Spies would produce microdots (small round images around 1mm in diameter) of important information, which would then be hidden, for example as the full-stop at the end of a typewritten sentence. To the naked eye, the information is invisible, and can only be viewed using magnification.

2.1.2 Uses of Steganography

The historical primary use of steganography is for message hiding. When methods such as cryptography are used, the output of these methods, such as the ciphertext, is often very noticeable. The reason for this is that these methods are designed to ensure that the message is unreadable to anyone other than the intended receiver, rather than hide its existence. Steganographic techniques, however, are designed to ensure that the message (or data) is

hidden and so observers do not know that the message exists. The hidden message or data may be encrypted before it is hidden for an added layer of security. The reason why this is useful is that in some countries it is illegal to even possess encrypted material. By using steganography this material can be hidden.

One modern usage of steganography is in watermarking. Watermarks are identifiers inserted into a piece of data (whether it be text, audio, video or an image), which can either be used to verify the data as genuine, or even identify a particular copy of the piece of data. An example is the release of an audio file containing a new song to reviewers, with the condition that it is not released. The file is watermarked with the identity of the reviewer. If the song is then leaked online, it will be very simple for the record label to access one of the illegal copies and extract the watermark, identifying who leaked the song. This process is however vulnerable, as will be discussed in the next section.

Steganography is increasingly being found to be used by terrorists and criminals to hide data and exchange information. For example, a child pornography distributor could hide images in the photos on an legitimate ebay listing to distribute them to their customers [11]. There has also been evidence of al-Qaeda storing information contained in text files hidden in videos [8].

Steganography is not just used by criminals and organisations. There are a large number of freely available steganography programs which can be used by home users to store sensitive information more securely, for example bank account details could be hidden in an image so when they are required they can be retrieved, but if someone was to gain access to the computer they would not be able to identify them (as they would be able to recognise, and possibly break, an encrypted file).

There is evidence of governments using steganography for both legal and illegal communication. There has been court cases in the US where accused Russian spies were found to have been using steganographic communication channels (specifically images) to communicate with their handlers [15].

There has even been proposals of using steganography as a means for communication within a botnet. Nagaraja et al. [23] proposed Stegobot, a botnet which uses steganography as the basis for its command and control (C& C) network. It uses JPEG steganogrpahy to hide collected data (such as credit card details and passwords) in images that are uploaded to Facebook. Bots are connected to each other via users on the social network. Once the images are uploaded, they are visible to all users connected to the uploading user. Bots on infected machines intercept any images that a user on that machine views, and extracts any information. This bot then hides the data in an image upload by a user on its computer, where it will be visible to all of that user's connections. The data eventually reaches the bot master via restrictive flooding. The estimated bitrate is around 20mb/month, which while not sounding like much can represent a large amount of personal information.

One particularly interesting case id that of printer manufacturers secretly printing microscopic dots onto all pages they print. These dots are arranged in a pattern which can be

decoded into the date and time the page was printed, along with the printers serial number [30]. It is claimed that this data is used by governments in criminal counterfeit investigations. This is an example of steganography where the data has been used to generate a pattern, and then this pattern has been printed at a microscopic level, following a similar (but much smaller) principle to microdots. There are a number of pieces of software that can be used to perform steganography. OpenStego [26] is an open source program for performing image steganography. It can hide any type of file within the image, and files are encrypted before they are hidden. OpenPuff [25] is a piece of freeware that can perform steganography on multiple file types, including audio and video (but not text), and can even split the steganography over multiple files so there are theoretically no data limits.

2.1.3 Types of Steganography

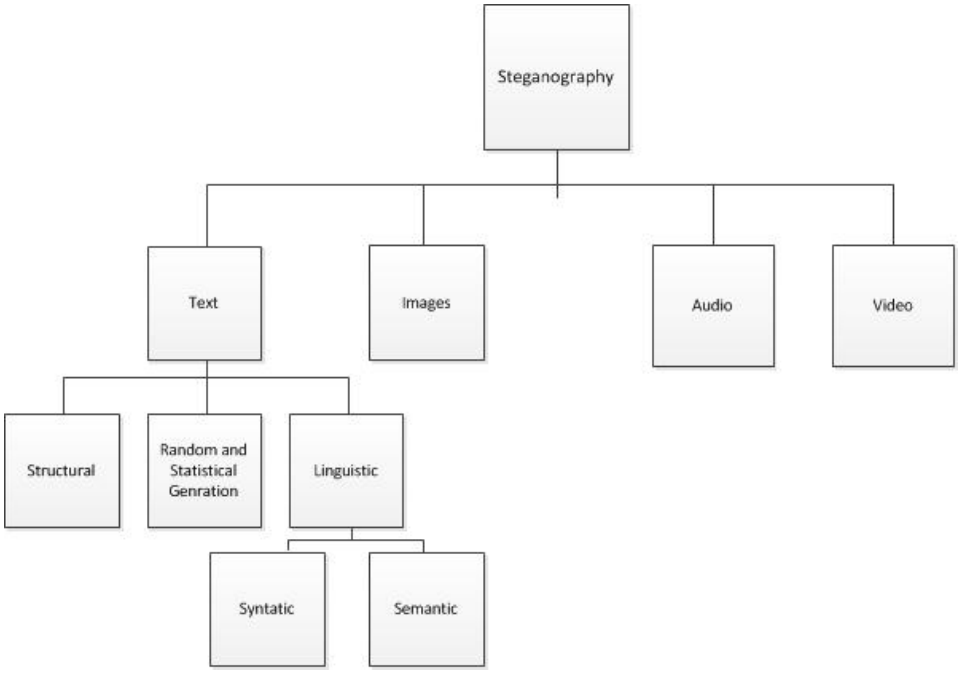


Figure 2.1: The different forms of steganography

Text

There are three mains forms of text steganography: structural, random and statistical generation and linguistic. Structural text steganography involves changing the physical structure of the text, for example by adding whitespace or increasing the line spacing. Random and statistical generation involves generating the cover-text either randomly, or according to some function on an input. Linguistic steganography, the focus of this project, involves

manipulating the syntactic or semantic properties of the existing text. These are discussed in detail in section 2.1.5.

Images

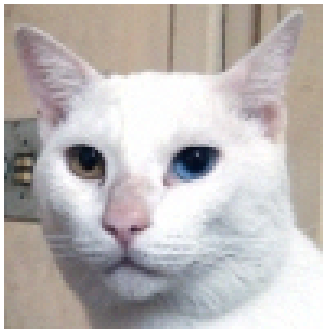


Figure 2.2: Example of image steganography. Image contains the text "Boss said that we should blow up the bridge at midnight." [39]

Image steganography can either be in the image or transform domain. Steganography in the image domain usually involves hiding data in the least significant bits at certain intervals in the image, for example one bit in each of the red, green blue values. In the transform domain, a transformation is applied on the cover image (such as the lossy part of a compression algorithm) and then the data is hidden. In JPEG compression this involves taking advantage of the discrete cosine transform. This means that the data is less susceptible to further compression algorithms. [22]

Audio

As with image steganography, data can be hidden using the least significant bit in the audio. Most commonly, the data is hidden in such a way that the data is inaudible to the human ear. For example, humans cannot hear a tone that immediately follows a louder tone, so this is often used to hide data as using the most significant bit can be used to help overcome audio compression, without the original file sounding any different to humans. Another possibility is to introduce a minute echo to sounds to hide data, the delay dictating the data being hidden [5].

Video

Video can be used to hide data in much of the same ways as with images by hiding data in each individual frame. A famous example of this is a video found on a laptop owned by an suspected al-Qaeda member. The video, at first glance, appeared to be pornography

but forensic investigators found that it contained 141 separate text files detailing terrorist operations and plans. [8]

2.1.4 Issues

Text

The three different types of text steganography are vulnerable to different attacks. Structural based methods can be destroyed if the text is copied or reprinted. For example, if the data is hidden through whitespace, but the cover-text is hand copied onto paper from a computer screen, then there is a very high chance that the data is lost.
Linguistic based methods are vulnerable to the text being changed by a third party attacker. A popular example with this relates to the prisoner's problem.

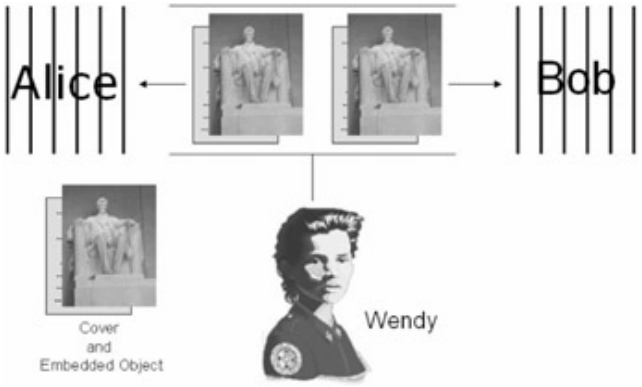


Figure 2.3: Representation of the prisoner's problem. [19]

In this scenario, the warden can intercept messages sent between prisoners and change words while still keeping the meaning of the text (as is done with synonym-based steganography methods). Doing this means that the correct data cannot be retrieved. An example of this, from the First World War is a cable-gram which read "Father is dead". The cable-gram was intercepted and the text changed to "Father is Deceased", which led to the response "Is Father dead or deceased?", which gave away that there was a hidden message. [16]

Images, Audio and Video

The primary vulnerability that these multimedia forms of steganography are susceptible to is compression. For example, in the case of an image, where data may be hidden in the least significant bit every k pixels, the image can be decompressed, distorted (for example by cropping) and then re-compressing [7]. Similar effects can affect audio and video. For example, in a audio file in the WAV format, data could be hidden in low volume or frequency sounds which humans cannot hear. If this file was compressed into the MP3 format, these

low frequency sounds are removed and so the data will be lost. The same effect can be had with noise reduction techniques, as the hidden data will cause noise in the medium. These can be done accidentally by a person who is unknowingly transporting the cover-text, or maliciously, for example by the warden in the prisoner's problem.
There are reports of some people, such as sound engineers and musicians, who claim they can hear the "high-pitched whine" which comes with hiding high-frequency information encoded in messages [1].

2.1.5 Text Steganography

As previously mentioned there are three main forms of text steganography, these will be described in more detail here.

Structural

Structural based text steganography involves manipulating the structure of the text in order to hide data. The structure includes the line spacing, word spacing, font size and anything similar. For example, one could use double word spacing to hide a 1, single to hide a zero. At a first glance, this will not be noticeable to the human eye. Margaret Thatcher reportedly used this method to aid in the prevention of press leakages of government documents [21]. A certain number of white spaces were placed in documents, the number being related to the cabinet minster to receive the document, so she could identify the owner of any leaked documents.

Random and Statistical Generation

There are a number of different ways to generate a cover-text to hide data. This can either be done randomly, so the words chosen are purely based on the data to be hidden, or statistical, so the words are chosen to match some statistical criteria. Statistical methods generally produce much more meaningful text that random methods. An example of a random method is Stego! [37]. This algorithm encrypts the data to be hidden using the JavaScrypt encryption algorithm, and then uses the Base64 result to select words from a random dictionary of 65536 words, one word for every two bytes in the data. For example, the word "hello" (encrypted with the key "123456") gives the result:
"Grayest fagoting slurped basophil lowlives foliums quality casefy. Sheols agonised, petiolar reassure grant. Autacoid ipecacs, rabbi saltires recoded. Branders serene clucks bellboys depsides. Conge."
The punctuation is added randomly. Obviously, this makes no sense whatsoever and any more than a quick glance will arise anyone's suspicions.
An example of a statistical approach is spammimic [38].This algorithm uses a "spam" grammar and the author's mimic function for context-free grammars (which generates text according to some statistical criteria) to create spam-like text which can be sent via email to

the recipient. For example, the word "hello" hidden with spammimic gives the result:
" Dear Decision maker , We know you are interested in receiving amazing intelligence . This is a one time mailing there is no need to request removal if you won't want any more . This mail is being sent in compliance with Senate bill 1625 ; Title 4 ; Section 302 . THIS IS NOT MULTI-LEVEL MARKETING ! Why work for somebody else when you can become rich as few as 55 DAYS . Have you ever noticed most everyone has a cellphone and more people than ever are surfing the web . Well, now is your chance to capitalize on this ! WE will help YOU decrease perceived waiting time by 200your business into an E-BUSINESS ! The best thing about our system is that it is absolutely risk free for you ! But don't believe us . Mr Ames of Massachusetts tried us and says "My only problem now is where to park all my cars" ! We are licensed to operate in all states ! We beseech you - act now . Sign up a friend and your friend will be rich too ! Thank-you for your serious consideration of our offer ! "
This is much more realistic than the Stego! example, as it appears almost exactly like a poorly written spam message. This approach takes advantage of the fact that 80 % of all email traffic is spam [9], and so discovering this email and extracting the bits among all of the other spam messages will incur a very high cost.

Linguistic

Linguistic steganography comes in two forms: syntactic and semantic. Syntactic text steganography involves altering the structure of the text without significantly altering the meaning or tone. Examples methods could be to alter the punctuation in a sentence. One proposed solution by Judge [14] uses spelling errors to hide data, for example spelling "is" as "iz". A correctly spelled word indicated a zero, a incorrectly spelled word a 1. This major flaw with this method is that it can be affected by unintentional spelling mistakes.
Semantic based steganography is the focus of this paper. This method involves replacing words with their synonyms in order to hide data. This will be explored more fully in the "Current Research" section below.

2.2 Steganalysis

Steganalysis is to steganography what cryptanalysis is to cryptography. The primary difference is that while with cryptanalysis the aim is to extract the original data from the encrypted message, the goal of steganalysis is primarily to detect if the object contains hidden data. It can be used to extract the message in some cases, although this is not always important and the steganography is considered to be broken if just the existence of hidden data is proven.

2.2.1 Adversary Models

When it comes to steganography, an adversary can either be passive or active.

Passive

A passive adversary, Eve, will intercept messages sent between Alice and Bob. They will be able to read all messages, but will not make any changes to the messages and will send them on intact to the receiver. They will run analysis on the message to try and discover hidden data. In the scenario of the prisoners problem, this will be the warden only reading the messages.

Active

An active adversary, Mallory, will again intercept the messages sent between Alice and Bob, but this time will tamper with the message in order to remove any possible hidden data. For example, if the message was an image or some audio, they could run compression or noise reduction algorithms to damage the hidden data. If a structural method of text steganography is used, they could remove any extra white space or make the line spacing uniform. They could perform synonym substitution. which will affect the data if a semantic method was used as different synonyms usually mean different data. In the case of the prisoners problem, this will be any case when the warden makes any change to the message.

2.2.2 Methods

There are a number of methods of steganalysis available to attackers. The primary and most common method is to use statistical analysis [12]. This can be of many forms. To perform this analysis requires the steganalyst to perform one of the six kinds of attacks.
The six main types of attack available to steganalysts [29]:

Stego-only Attack

The attacker only has access to the stego-object. The attacker would need to perform statistical analysis on the object. For example, structural text steganography can be easily detected simply by searching for extra spaces, unusual line spacing and so on [1]. For semantic methods, this could involve using analysis of corpora to analyse the occurrences of words within text and then examining the suspicious message to see if there are any words which do not appear frequently in the corpus with the surrounding words. This may indicate that a word has been replaced with a less commonly used synonym. This method could also involve the construction of large context free grammars which will easily be able to identify unusual elements in text.

Known Cover Attack

The attacker knows the original cover-text and the stego-text and can detect pattern differences between them. This can be used to discover patterns which can be identified in future suspect objects [13].

Known Message Attack

The attacker only knows the message. This attack involves the analysis of know patterns the correspond to hidden information. This is very similar to the stego-only attack in difficulty.

Chosen Stego Attack

The attacker has access to the steganography algorithm and the stego-object. This can be used to discover typical outputs of the algorithm, which can then be identified later on in future objects.

Chosen Message Attack

The attacker uses an steganography algorithm to generate stego-objects. He then analyses these and uses this information to identify patterns that can indicate the usage of the algorithm in future objects.

Known Stego Attack

The attacker knows the algorithm, the original object and the stego-object. With this information the attacker would have little trouble reverse engineering and discovering exactly what the algorithm does to the original object. He could then use this information to easily detect objects which have had the algorithm applied to them.

These six methods can be used for automatic analysis. In some cases a human will be able to perform the analysis. For example, with linguistic text steganography a professional linguist would be able to spot any unusual semantic or syntactic features. And as previously mentioned there are reports of audio specialists being able to here the data hidden in audio.

2.3 Current Research

Shirali-Shahreza and Shirali-Shahreza [32] proposed a novel method of synonym-based text steganography which makes use of the differences between American English and British English, for example the difference between the words "football" and "soccer". The basic idea is that to hide bits, whenever an word has a different word for American and British English is found, the word is replaced by either its American (for a 0) or British (for a 1) alternative. As with the other mentioned algorithms, they report a very low bitrate, but

point out that compared to standard synonym-based steganography methods there is very little chance of the semantic meaning of the cover-text being lost and there is a low risk of detection, in particular in countries where English (neither American or British) is not the native language as the mix of American and British English words will not attract as much attention.

Shirali-Shahreza and Shirali-Shahreza have also published a second method of text steganography [33], which makes use of the abbreviations used in "test speak". For example the replacement of the word "sorry" with "sry" or "see" with "c". Using the same principle as the American-British method, they use the full word to hide a '0', and the abbreviation to hide a '1'. This method is effective against detection if it is used in text where abbreviations are common (such as SMS messages or instant messaging), but will be easily detected if it is used in a text medium where abbreviations are not common.

Igor Bolshakov [2] produced an algorithm for synonym-based substitution using the theory of transitive closure to build his synsets (sets of synonyms). The transitive closure was applied to overlapping sysnets which removes ambiguity when decoding. For example, the overlapping synsets for the words "paper" and "composition" are {'authorship', 'composition', 'paper', 'penning', 'report', 'theme', 'writing'}. The main issue with this approach is that all of the words in this set need to be considered. The set could be very large, and some of the words may not be appropriate for the word in most cases.

Topkara et al. [35] have proposed a synonym substitution based algorithm for the specific purpose of watermarking text documents. The synonyms to be chosen are decided upon by calculating their distortion effect (the effect of making the substitution on the "value" of the document) with a goal of maximising the distortion (within an acceptable limit), and also their resilience to detection. More ambiguous words, such as "bank" which has multiple different senses, are preferred as they are less likely to not fit into the text.

Chang and Clark's [4] approach is to use the idea of synonym substitution and vertex colour coding. The algorithm generates a coloured graph of the synonyms of the word and then uses this to choose the synonyms, which are tested using n-gram frequency counts from the Google Web n-gram corpus [6]. The removes any synonyms which would fail statistical analysis. The synonyms are chosen using this graph, and the synonym that is chosen depends on the data to hide. To retrieve data this graph is again constructed for each word, which should be the same graph as for the original word, and then the word's position in the graph determines the hidden data. They achieve a bitrate of just over one bit per standard newspaper sentence, which is roughly the aim of the algorithm in this project.

Chapter 3

Design

3.1 Algorithm

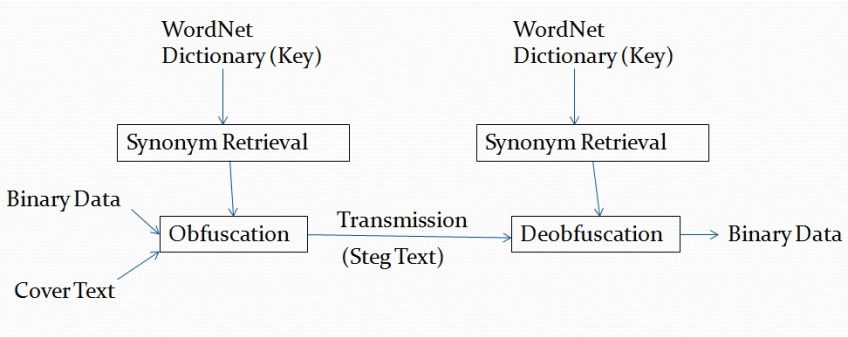


Figure 3.1: Figure showing structure of algorithm

3.1.1 Dictionary and Corpora

Dictionary

The dictionary that is used for the algorithm is the WordNet dictionary, produced by Princeton University in the US [20]. The dictionary contains around 150,000 words, organised into sets of synonyms, called synsets. A synset consists of the set of synonyms, a brief description (definition) of that sense, and in some cases one or two example sentences showing a typical usage of one of the words in the synset. There are around 115,000 synsets, some containing just a single word and some containing more than 8 synonyms. A search for a word in the dictionary database will return all synsets which contain that word, separated for the different word types (noun, verb, adverb and adjectives), and sorted by the frequency that they appear in some sample texts so the most common sense of a word is displayed first.

Searches for words that appear in the same synset return the same synset. It is this synset structure that makes WordNet ideal for use in this project.

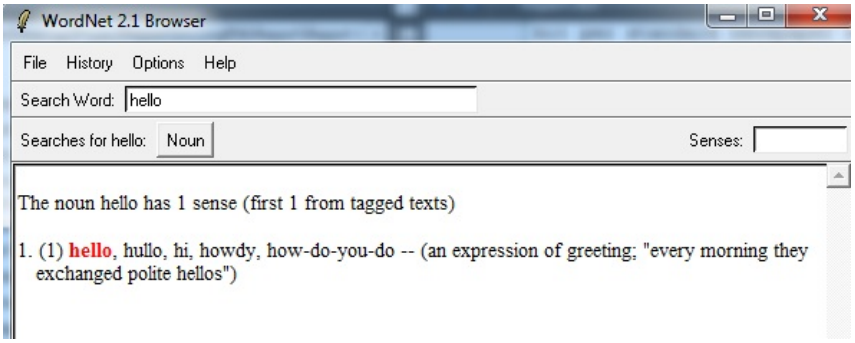


Figure 3.2: Screenshot of WordNet application after querying the word "hello"

Corpora

There are two sets of corpus data which are used in the algorithm. The first is from the British National Corpus (BNC) [3], and shows the frequency of a subset of words (along with their part-of-speech tags) which are found in dialogue (a subset is used as the full list of words is almost a million words long) [18]. The second set is taken from the American National Corpus (ANC) [28], and lists bigrams of words and their frequencies. The bigram data is limited to bigrams which have a frequency of more than 4, as below this the frequency is too low and also it limits the amount of data that needs to be searched (the full set is over 2 million in size, with this limit it is approximately 250 thousand).

3.1.2 Synonym Retrieval

Synonym retrieval is perhaps the most important aspect of the algorithm. If the synonyms are not retrieved properly, it would be difficult to ensure that data can be deobfuscated as there is guarantee that the synset you generate when deobfuscating is the same as when you obfuscated. A bad synset will also include words which are only very loosely synonymous with the original words, which will affect the robustness of the algorithm.
The first task in retrieving the synonyms for a word is to decide which of the four word types (noun, verb, adverb or adjective) the word is. In some cases this is simple as the word can only be of one type, but in many cases this is not the case and the word can be of two or more of the possible types. If the wrong type is chosen, then the synonym chosen may not make sense. To choose the most likely word type, the word frequency data from the BNC is used. The tags used in this list are much more detailed than in the WordNet database, so the word may appear in the list multiple times with many different tags all representing the same tag. When a word is searched in this list, the total frequencies for noun, verb adverb and adjective are all found, and the greatest frequency is used as the correct tag for the

word. If the word is not in the list the word type with the most senses in WordNet for that word is chosen.
To enable the same (or a close as possible) set of synonyms to be returned for both the original word and the word that replaces it, as large a portion of the entire set of synonyms needs to be retrieved. This is done by retrieving the set of synonyms for all senses of the word that has been entered (whether it is to obfuscate or deobfuscate), using the word type found by the process described above. All of these senses are added to a list. The algorithm then goes through each word in this list in turn, and adds all senses for the current word type for each of those words, ignoring words that are already in the list. This then provides a list of a large portion of the words complete set of synonyms. The algorithm could repeat this process again, but with each level of abstraction from the original word the returned synonyms are less likely to be actually related to the word itself.

Data: word
Result: synset
synset = getsynonymsfromdictionary(word);
for $i \leftarrow 0$ **to** *synsetsize* **do**
| synset.addall(getSynonymsfromdictionary(synset[i]);
end
for $i \leftarrow 0$ **to** *synsetsize* **do**
| synset.setfrequency(getfrequency(synset[i-1], synset[i]);
end
synset.sort();
return synset;

Algorithm 1: Synonym Retrieval Pseudo-code

During this process, any senses which contain a single synonym are ignored, as well as any synonyms that are composed of two or more words, contain hyphens or are less than 3 characters in length. The words with spaces are ignored because it is impossible to know that they represent one synonym when deobfuscating. The words of less than 3 characters are ignored because in most cases they are either not in the dictionary, or they are in there representing an abbreviation for a name. For example, a synonym of "hello" is "hi", but "hi" also belongs in s synset with "Hawaii", which would obviously not be appropriate as a synonym as it has a completely different meaning to "hello".
Finally, this list is sorted according to the frequency of the bigram of that word and the word before it. The bigram frequencies are taken from the bigram frequency data from the ANC. This sorting ensures that the more likely synonyms to appear in the text are chosen above less likely ones.

Punctuation

Punctuation is not removed from the inputted text. A word such as "don't" will be searched as such. The reason for this is that it enables the algorithm to keep punctuation in the

outputted text. It only actually affects a small number of words, so even though it will limit the bitrate slightly it will not reduce the effect on the quality of the sentence. If the punctuation was removed and reinserted after word replacement, the punctuation may not fit the replacement word.

3.1.3 Obfuscation

Obfuscation is the term used to represent hiding data in the text. To obfuscate bits is a relatively simple process. Two components are required, the cover-text and the bitstream (the list of bits that need to be hidden). For each word in turn in the cover text, the full synonym list is retrieved using the process described above. It is then a simple case of choosing the appropriate synonym to be returned. If the next bit in the bitstream is a zero, the first element is returned, if it is a one, the second, and if the next two elements are 10, the third (representing a 2, the value of binary 10) is used (this helps to increase the bitrate).

Data: Synset, bit to hide
Result: Chosen Synonym
if *synset size less than bit* **then**
| return synset element at position bit;
end

Algorithm 2: Obfuscation Algorithm Pseudo-code

Before the word is actually replaced, two tests are performed. The first is to attempt to deobfuscate the word and see if the original bit(s) is returned, if they are not then the word is unsuitable and so the bit(s) is not removed from the bitstream and the original entered word is used rather than the replacement synonym. The second test is to check the quality of the replacement using the quality test described below. If the quality is below a threshold, the bit(s) are not removed from the bitstream and again the original word is used. If the word is the first in the cover-text, then the quality check is instead the frequency for the most likely word type, taken from the BNC word frequency data. If the replacement word passes both of these tests, then the replacement word is returned and the hidden bits removed from the bitstream so they are not added again. It is worth noting that in many cases the replacement word can be the original word itself. if the bitstream is empty when obfuscation is attempted, the algorithms always tries to hide a 0 in the word. This is due to the fact that the deobfuscation algorithm always expects there to be data hidden in the text.

3.1.4 Deobfuscation

Deobfuscation refers to extracting hidden data from the text. To extract the hidden data from a word, the first step is for the word to pass some tests to limit the number of false positives. The primary test is to perform the quality test with the word before it. If the

quality is found to be above the threshold (set to the same as for obfuscation), then the algorithm proceeds to attempt deobfuscation. Only words which have passed this quality test can contain data, so if it fails you know that no data is hidden in the word.
To deobfuscate a word, the synonym list for the word is found as previously described. This should return as close to the synonym list of the original word as possible. The algorithm then retrieves the position in this list of the word to be deobfuscated, and this position is taken to be the hidden data (with the first and second positions taken to be 0 and 1 respectively, and the third position 1,0). If the word is not within the first three positions then it is ignored and it is assumed no bits are hidden (as it would fail the deobfuscation check when obfuscating).

```
Data: Synset, word
Result: Hidden bit(s)
for i ← 0 to synsetsize do
    if synset[i] = word then
        return i;
    end
end
return null;
```

Algorithm 3: Deobfuscation Algorithm Pseudo-code

3.1.5 Quality Checks

The quality test uses the bigram data from the American Nation Corpus [28]. The tests are performed on the word that is being processed by locating the frequency in the bigram data by searching for the bigram previousword-current. If the word is the first in a sentence, you cannot test with the word before, so in this case the search is for the bigram current-nextword. If the bigram is not in this list it has a frequency less than 4, so the returned quality is zero, else the returned quality is the frequency.

3.2 StegChat

The StegChat application will be a prototype demonstration application for the described synonym-based text steganography algorithm. The application will be a web based application which will facilitate the continuous "chat" between two users. When supplied with data, the algorithm should attempt to hide as much data as possible without compromising the quality of the cover-text messages. The user who receives the messages should be able to easily retrieve the hidden data, preferably automatically.
The application will take the form of a web application, so the entire application will be usable within the users browser. The site will be built using Java servlets and jsp pages as much as possible for two reasons; firstly Java is my primary language and secondly by keeping the algorithm code within Java it will be easy to produce a local version of the application if required.

3.2.1 Basic Structure

The basic structure of how the program should operate can be seen in figure 3.3 below.

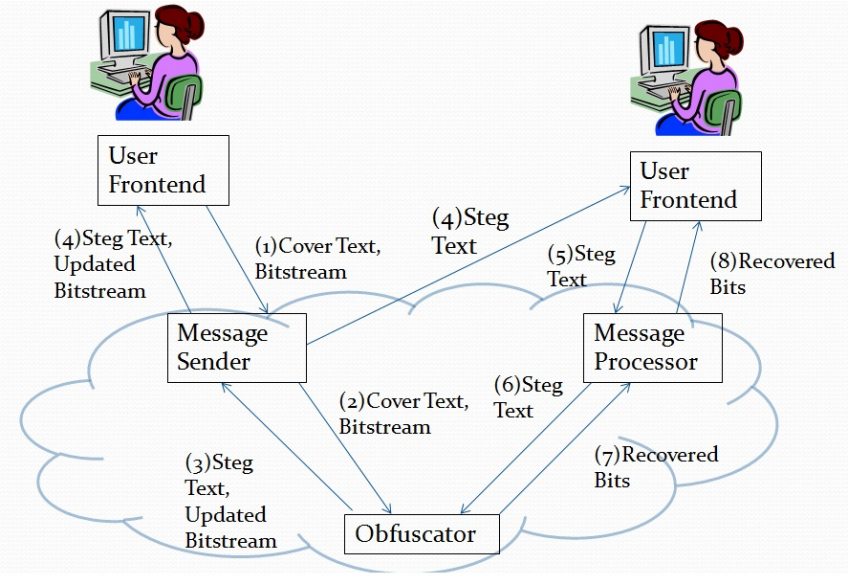


Figure 3.3: Diagram showing structure of chat application

The user enters the text to send into the application, along with the data to hide, which is passed to the message sender (1). This then calls the obfuscator algorithms with the text and bitstream (2), which returns the updated bitstream and the stegtext (3). These are returned to the user, with (the text only) also being sent to the receiver (4). When the receiver receives a message, they pass it to the message processor (5), which calls the obfuscator to deobfuscate (6). This returns the hidden data (7), which is passed back down to the user(8) to be displayed on the screen.

Chapter 4

Implementation

In this section I will discuss the technical details involved in producing the algorithm an StegChat test application. A structure diagram showing the classes used during key processes can be found as Appendix B (This diagram does not contain every single class. Exceptions include object classes which are explained in sufficient detail).

4.1 Platform

The Google App Engine [10] was chosen as the platform on which to develop the application. The App Engine is Google's cloud-based application hosting service. There were a number of reasons why this was chosen. The primary reason is that Java is the authors first language, and it is one of the App Engine's three supported languages (the others being Python and Go). It also provides access to the High-Replication Datastore, a database-like service which has very fast access and promises no data loss (due to the high replication aspect). Another (and perhaps most important) reason why it was chosen as for small usage applications it is completely free (which is ideal for development). It does, however, have a few limitations:

- One minute request time - All requests have a limit of one minute for them to be returned. This means that any processing must take under one minute, and so long sections of text can not be processed (although in the case of a chat application this is not an issue).

- No support for outgoing TCP connections - This limits the ability of any application to connect to external servers using TCP. In the sense of a chat application, this means that the use of an external XMPP service (such as Facebook Chat) is limited. The overcoming of this issue is discussed later in this chapter.

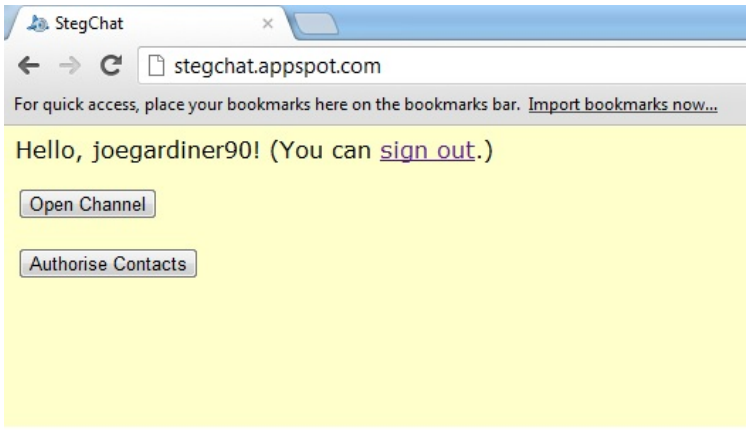


Figure 4.1: Home Screen. User signed in using Google User Service. User clicks "Open Channel" to go to chat screen

4.1.1 Resources

The application was developed using the Eclipse IDE, using the Google App Engine plugin. The website was developed using Chrome and Firefox.

4.2 Dictionary Storage

One of the biggest challenges with developing the system in a technical sense was the issue of how to store and access the WordNet library. For a local Java application, this is very simple. You install the WordNet database on your computer, and then there are a number of libraries that can be used to access the database by pointing them to the database files. The database files are simply large text files, ranging from a few kilobytes in size to fourteen megabytes. There are two primary types, index files, which for each word list the senses which that word belongs to, and data files, which contain the senses. There is one of each of these files for each of the four word types. The data files are accessed via offset values, which are contained in the file, and the index file simple lists these offsets.
When it came to transporting this model of a library and database files onto the App Engine, a number of issues arose. Firstly, the files had to be stored on the App Engine. The App Engine provides a file store called the BlobStore which is primarily used for hosting content such as video or images to be delivered to users of the application. There is no upper limit to the file size, but there is a maximum fetch size of one megabyte (anything larger has to be fetched sequentially). With this being the only viable way to store the files themselves on the App Engine, this also caused a problem for the Java libraries used to access the database. Because of this, it was decided that, as the files were relatively easy to parse and process manually, that libraries would not be used. This was tested with the application,

with the files fetched and read by a simple BufferedReader on a FileInputStream. While this worked, it was extremely slow. For example, the word "Hello", when searching only for noun senses, would take around 45 seconds, and would fail if the other word types were also searched (due to the one minute request limit). This is due to the fact that to search the database each file had to be fetched and read line by line and processed. The noun index itself has over 100,000 lines, which while not a problem in local testing, is extremely slow on the App Engine (due to the requirement to fetch the file as it is being read).

The next option was to make use of the App Engine's high-replication datastore. This is a database-like feature, hosted on the cloud with the application. It is high-replication in the sense that your data is replicated among a number of data servers, reducing the risk of data loss. It has a very fast access time for application running on the App Engine due to a very clever indexing and caching scheme. The main issue with the datastore is the quotas imposed on free applications. There is a limit of 50,000 write operations a day (with each insert operation requiring 4 writes and 1 read). An attempt to insert the WordNet database as objects (with each line of the index file as an Index object and a similar structure for the data files) hit the quota within second of inserting the first file, which was obviously an issue. To overcome the problem, the datastore bulkuploader function was used. This is a Python application which enables the insertion on csv and xml files straight into the datastore using the remote api. This required a Python instance of the application to be run alongside the full Java version, which enabled the Python bulkloader function to be used to access the live datastore. All of the files were converted into csv files using a simple java program, included as part of this project. All of the four index files were included as one single csv file of a "MasterIndex" type, which enables all of the four types to be retrieved at once. The data files were kept separate, so that they only needed to be searched if they exists in the index. The MasterIndex file has three fields, the word, the offsets of its senses (as a string which is split for processing) and its type (noun, verb etc). The data files have two fields, the offset and the synonyms (also as a string which is split for processing when required).

The basic method of searching for a word in the datastore version of WordNet is to first query the MasterIndex table, which return all the sense offsets for every type of the word. For each word type, if senses exist they are queried in the appropriate data table, using the list of offsets as a parameter and the "IN" filter operator. All of these senses, for all types, are then formed into a "Word" object, which represents all of the senses(and therefore synonyms) for the entered word. These word objects are then used to build the full sense list for the word to be used in the substitution.

The two sets of corpora data were also inserted into the datastore in the same way.

4.3 Chat Screen

The chat screen is produced using a combination of a Java Servlet Page (JSP) file, plus a number of java servlets. The main page is chat.jsp, which makes AJAX requests to servlets to perform tasks, include generating elements to display on the page. AJAX is used as opening the channel is fairly CPU intensive, and so AJAX requests mean that it only has to be done once as the page does not have to be reloaded, closing the channel. It also means that any received messages are not lost if the page is reloaded.

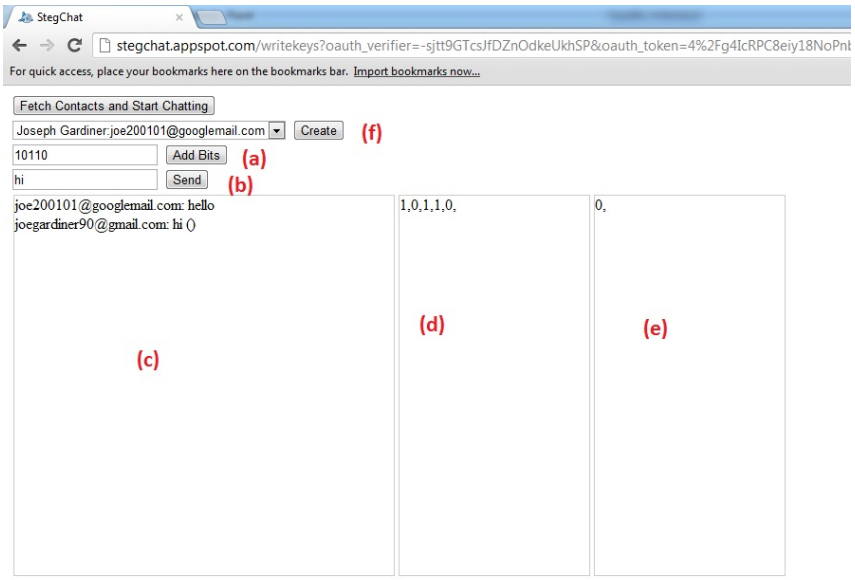


Figure 4.2: Full Chat Window. Components: (a) Form for adding bits (b) Form for sending message (c) Chat Log (d) Bitstream (e) Recovered bits (f)Contact Selection

4.3.1 Contacts Authentication

One of the extra features that was included to make setting up a chat with another person easier is the ability to fetch all of contacts from the users Google Contacts list. Google provides the Google Data API for accessing various pieces of data. There is one overall API, and then this is used in conjunction with more specialised APIs to retrieve the desired data, such as contacts or Googls Docs documents.

There are a number of steps required in fetching the contacts data. The data is retrieved by fetching a feed for the user using a specific URL and a ContactsService object (from the contacts data API). The first is for the user to authenticate the application to access their contacts. This is performed by generating an access token using OAuth authentication; the application forwards the user to a page provided by Google which asks the user to authenticate the application.

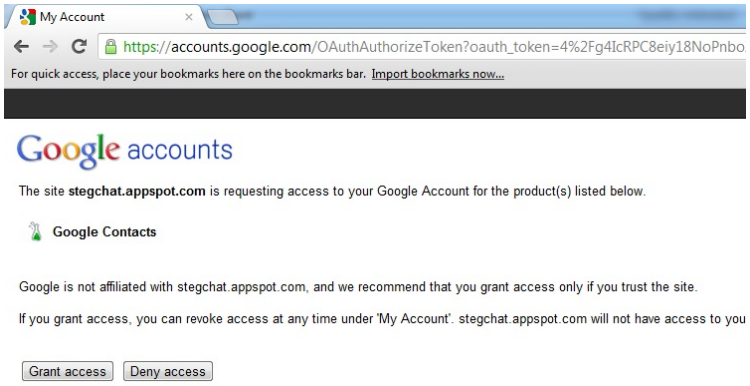


Figure 4.3: Authentication for Google Contacts

This returns the secret token key, which is used with the applications token to access the contacts list. These two values are stored, along with the username, as a custom SessionKeys object in the memcache. The memcache is a short term memory primarily used for storing the results of commonly used queries from the datastore. When the memcache is full, elements are deleted to make way for the new data. The SessionKeys objects are set to expire after one hour, so each time a person visits the site they have to re-authenticate , but they will not have to perform this step again within an hour (unless the keys are pushed out of the memcache).

To fetch the contacts list the following code is used:

```
URL feedUrl = new URL("https://www.google.com/m8/feeds/contacts/
    default/full");
ContactFeed resultFeed = myService.getFeed(feedUrl, ContactFeed.
    class);
int count = resultFeed.getTotalResults();
URL feedUrl = new URL("https://www.google.com/m8/feeds/contacts/
    default/full");
Query myQuery = new Query(feedUrl);
myQuery.setMaxResults(count);
myQuery.setStringCustomParameter("sortorder", "ascending");
ContactFeed resultFeed =  myService.query(myQuery, ContactFeed.
    class);
for (ContactEntry entry : resultFeed.getEntries()) {....}
```

The code above first fetches the results feed using the feedURL, and gets the total number of contacts. It then produces a query on the feedURL class so that the results are sorted. This

has to be done in two steps because fetching the result feed doesn't always contain all of the contacts, you need the second query, with the count, to fetch them all. The query is called in the feedURL, which is the URL of the contacts feed, which returns a ContactsFeed object. This can be used with an iterator to process the results, in this case (though removed from the above snippet) the results, if they contain an email address, are used to create an html form containing a dropdown box which is returned to the chat.jsp page to be displayed, and is used to select a contact to chat with.
In the scope of this demo application, there is currently no way to know if a contact is online or using the application. This is not provided by the contacts API or the Channels API (as described below). It would have taken too much time to manually produce this feature for a prototype and make it fit with the App Engine's strict quotas, but if more time was available then it could be implemented. Also, the users both have to open channels to each other for them to receive messages.

4.3.2 Channels API

To facilitate the chat communications the original plan was to use the App Engine XMPP library to provide the communication. There were two main issues with this. Firstly, the XMPP addresses had to be of the form "username@stegchat.appspotchat.com" and an existing XMPP account, such as Facebook chat or Google talk could not be used due to the restriction on outgoing tcp connections. Secondly, all messages have to be received by a specific handler, no matter what address they are sent to, and then somehow passed down to the appropriate users. The first attempt to do this was to use the memcache (as is used with the contacts authentication). The problems mentioned before could cause problems for this (messages being dropped before they are retrieved). The datastore was not an option as to write the messages to the datastore for any more than a few users would run over quota very quickly.
Fortunately, Google offers the Channels API for providing a method of communication between instances of the application, for example for sending data to a user from a backend process. They are essentially a local version of UDP sockets, as in a channel is opened using a token generated from a string (the address). The channel itself is opened via JavaScript, and it is up to the user to decide what happens when certain event occur, such as receiving a message:

```
1 <script type="text/javascript" src="/_ah/channel/jsapi"></script>
  </head>
3 <body>
  <script>
5   var channel = new goog.appengine.Channel('<%=token%>');
    var socket = channel.open({
7         onopen : function() {
```

```
        },
        onmessage : function(message) {
                var messagesNode = document.getElementById(message
                    .data.split(":")[0].concat("messages"));
                var messageNode = document.createElement('div');
                messageNode.appendChild(document.createTextNode(
                    message.data));
                messagesNode.appendChild(messageNode);
                processMessage(message.data.split(":")[0], message
                    .data.split(":")[1]);

        },
        onerror : function(error) {},
        onclose : function() {leave();}
});
```

The code above takes the received message, splits the text and extracts the sender address and message, and then adds the message to the appropriate chat box and sends the message to be processed via the processMessage function which calls a servlet via AJAX.
To token is passed to the page as a parameter when the page is loaded from an CreateChannelServlet servlet, which is called when the user decided to go from the homescreen to the chat screen. This servlet uses the Google user service (provided as part of the App Engine API) to get the email address of the logged in user, and use this to generate the token for the channel. This is added to the request as a parameter and sent to the chat.jsp page where it is used as above to open the channel.

```
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
ChannelService channelService = ChannelServiceFactory
                .getChannelService();
String token = channelService.createChannel(user.getEmail().trim()
    );
```

Messages can then be sent to this user simply by using the email address as the destination when creating a ChannelMessage object (the Channels API equivalent of a UDP packet):

```
ChannelService channelService = ChannelServiceFactory
                                        .getChannelService();

ChannelMessage toSend = new ChannelMessage(destEmail, userEmail
                                        + ":" + output);
channelService.sendMessage(toSend);
```

4.3.3 Chat Box

When the user selects a contact that they wish to talk to, a chat box is produced using a servlet to generate the code. An AJAX request is made to the CreateChatBox servlet, which requires the selected contacts email as a parameter. This then produces all of the components required to chat: two simple text forms for inputting bits and messages to send, a text element for a chat log, a box for displaying the current bit queue, and a box to display any received bits. Each element's name includes the contacts email,for example *email*sendbits or *email*messages

Once the servlet has created the chat box it is sent in the response back to the chat.jsp page where it is added to the main div element as below:

```
xmlhttp.open("POST","/createchat",true);
var email = document.createChat["contactSelect"];
xmlhttp.setRequestHeader('Content-Type', 'application/x-www-form-
    urlencoded');
xmlhttp.onreadystatechange= function ()
 {
          if (xmlhttp.readyState==4) {
                   if (xmlhttp.status == 200) {
                      var chatsNode = document.getElementById('chats
                          ');
                      var chatNode = document.createElement('div');
                      chatNode.innerHTML = xmlhttp.responseText;
                      chatsNode.appendChild(chatNode);
                   }
         }
 }
xmlhttp.send("contactSelect=" + email.value);
                                                    }
```

4.3.4 Message Sending and Receiving

Messages to be sent are typed into the send text box on the chat form for the contact which is to receive the message. The message, along with the current bitstream and the destination address, is sent to a message sender handler via an AJAX call. This handler calls the obfuscator class, which performs the steganography, and sends the output via the channels api to the receiver. The result, with any words which contain bits in upper case, is returned to the sender handler, along with the updated bitstream, which is then written to the response and sent back to the chat form to be displayed. This means that there will be a delay between the message being entered by the user and the result being outputted to the

screen (although there is almost zero delay in the message being received by the recipient). When a message is received, it is first added to the chat box so it is displayed on the screen. It is then sent to a message receiver handler, which as with sending calls the obfuscator class for deobfuscation. This time the message is not sent, and all that is returned is the bits that have been recovered. These are sent back down to the user and displayed in the recovered bits box in the chat form.
If there are no bits in the bitstream then the algorithm by default attempts to hide a 0 in every word when obfuscating. This is due to the fact that the deobfuscation method expects data to be contained in every word, and so if there was no data it would be a lot less predictable.

4.4 The Algorithm

4.4.1 Algorithm Operation

The algorithm operates within a number of standard Java Classes. There are two main classes, one for processing the text and generating the synsets (the WordQuery class), and one for performing the actual obfuscation (the Obfuscator class).
The WordQuery class contains a number of methods for processing the text. The primary method, doQuery(), takes the cover-text, bitstream and the operation to be performed as parameters and then converts the text and bitstream into two separate ArrayList objects. The method then either performs obfuscation or deobfuscation depending on the operation parameter. The synset is then generated as described in sectiopn 5.4.2 below.
The list of returned FrequencyString objects is passed to the Obfuscator class. This class simply returns the element in the list at position 0 for bit 0, at 1 for bit 1, and at 2 for the bits 1,0. The deobfuscation method also in this class, also takes the in-putted word as a parameter and returns its position in the list.
The order of operation in obfuscation and deobfuscation is slightly different. To obfuscate, the synset is generated, and then the obfuscation method called. The ability to deobfuscate is then tested by generating the synset for the result and passing it to the deobfuscator method in the Obfuscator class, and if the bit that is retrieved is the same as the one that was hidden then the test has been passed. If this test is passed, then the quality test is performed using the code below:

```
int quality = 0;
if (i == 0 && input.length > 1) {
        quality = qualityTest(result, input[1]);
} else if (i > 0 && i < input.length - 1) {
        if (input[i - 1].trim().endsWith(".")) {
                if (i < input.length - 2) {
                                quality = qualityTest(result,
```

```
8                                   input[i + 1]);
                } else {
10                          quality = qualityTest(
                input[i - 1], result);
12            }
    } else {
14          quality = qualityTest(
                input[i - 1], result);
16    }
} else if (i > 0 && i == input.length - 1) {
18        quality = qualityTest(input[i - 1],
                                  result);
20
} else {
22        quality = 100;
}
```

This basically calls the getQuality() method, which queries the bigram data and return the frequency. In most cases the parameters are the previous word, then the result of the obfuscation, although for the first word in a sentence the parameters are the result and the next word. If the quality is above a certain value, then the obfuscation is finalised and the appropriate number of elements removed from the bitstream. The input list is updated with the new word, so the quality checks always take into account replacement words.
With deobfuscation, this quality test is performed first. If the result is above the threshold, then it is assumed data is hidden so the synset is generated and deobfuscation is attempted (only if a synset is found). If the result of deobfuscation is not 0, 1 or 2 then the word is ignored. If there is a valid result, then the bits are added to the recovered bits list to be returned to the user.

4.4.2 Synonym Retrieval

For each word in the input, if it is at least 3 characters in length a method, getFullSet() is called which assembles the synset using the getWord() function. This function queries the MasterIndex table, as described above, and then calls four individual classes for fetching each of the noun, verb, adverb and adjective senses (only if there are any listed in the index file). These classes simply perform a query on their appropriate data table and return the synonyms as a two-dimensional ArrayList of String objects. These are stored in a Word object, which is used by the getFullSet method to build the initial synset. It then repeats this process for each of the words in the synset, and adds their synonyms to the set, ignoring duplications. The quality of each of these words is found using the getQuality method (using the word before and the current word as parameters), the result of which is stored, along with

the word, as a FrequencyString in a list, which is then sorted using the Collections.sort() method.

4.5 Known Issues

There are a few known issues with the system that were uncovered during testing.

Sentence Length Limit

Due to the one minute request time sentences that contain words which have a large number of synonyms, or a large proportion of words which are possible carriers, can take longer than one minute to process and hence cause a "DealineExceededException". This means that the message is not sent or returned to the screen. This problem can partly be overcome by increasing the clock speed and memory available to the application (which is by default low), but this uses the resources at a higher rate, affecting the quotas.

First Request Failure

One of the features of the App Engine is that if a process is not used for a certain amount of time, it is shut down and has to be restarted. This is not a problem with regular use of the application, but if the application is not used for some time (measured in hours) then the first request to the server usually fails even for a simple single word such as hello due to the time required to restart the processes. The page has to be reloaded for the application to work after this. Due to this being an issue with the App Engine itself there is no obvious fix, but would not be a problem if enough people use the application regularly.

Chapter 5

Evaluation

There will be two forms of evaluation performed, statistical and user. The statistical evaluation will be used to determine the bitrate and the algorithms effectiveness against statistical steganalysis, while the user testing will allow humans to evaluate the output and test its effectiveness against human steganalysis.

5.1 Setup

To enable a full evaluation of the algorithm to take place, the system had to be adapted to accept larger text strings by removing the one minute processing time on the App Engine. To achieve this, the algorithm code was taken from the App Engine program, and the datastore was replaced with the actual WordNet dictionary files, accessed using the Java API for WordNet Searching (JAWS) [34]. The corpus data is loaded into the system and stored as ArrayList objects. This allowed the algorithm to provide identical results to the live web-based version, but accept any length of text. The evaluation program removes all of the features from the web chat application and runs through the command line, with the only remaining feature being the steganography itself.
The text is processed with a random bitstream. The result of the obfuscation algorithm is returned as an ObData object, which contains the word after processing and any bits hidden. The final element in the list (after the obfuscation algorithm is used) is a string containing the full text output. When deobfuscating, the result is identical, except the ObData objects in the list contain the deobfuscated word and any bits found. The lists created by obfuscation and deobfuscation are then compared to generate the test data.

5.2 Test Data

To test the algorithm text from a number of sources will be used to compare the effectiveness of the algorithm on different styles of writing. These will be:

- Online News Articles
- USENET Postings
- Academic Paper Extract
- Text From a Piece of Fiction

Where the source (such as the news article) contains headings or extra pieces of text (such as extracts), only the main body of the text will be used.
The data to be hidden will be randomly generated bitstream of around 2000-3000 bits. This data will be used for all pieces of text so a direct comparison can be made. The tests will be repeated with a second set of random data as the bits in the bitstream can make a different to the output, in particular the bitrate.
All the sample text and data files along with their outputs for the two bitstreams, can be found on the CD.

5.3 Statistical Evaluation Criteria

The text will be tested in three ways:

- BitRate - For each piece of text the bitrate per 10/100 words will be calculated.
- Quality Test - For each word in the text the quality will be calculated before the data is hidden and afterwards to discover the impact on the quality of the text. The quality will be calculated using the bigram data in the same way that the quality is tested during obfuscation in the actual algorithm. The quality will also be given as an average per word in the text. Two values will be provided, total score for the text and the average per word.
- Correctness - This will be calculated by calculating fie different values.
- User Survey To test the robustness of the algorithm to human inspection a short survey will be used containing sample output from the above tests.
  - False Positives - The number of words in which bits are found by deobfuscation which do not contain any hidden data.
  - False Negatives - The number of words in which data is hidden but not recovered by deobfuscation.
  - True Positives - The number of words in which data is hidden and found.
  - True Negatives - The number of words in which data is hidden and no data is found.
  - Error Count - This will be a count of any situation where bits are hidden and bits are found, but the recovered bits are not the bits that were originally hidden.

5.4 Statistical Evaluation Results

5.4.1 News Article

The news article used was taken from the BBC website, on the subject of the London 2012 Paralympic Games [24].

Results

	Bitstream 1	**Bitstream 2**
Words	857	857
Total Bitrate	64	60
Average bitrate (/10 words)	0.7	0.7
False Positives	10	13
False Negatives	0	0
True Positives	57	55
True Negatives	790	789
Error Count	0	0
Original Total Quality	1340403	1340403
Original Average Quality	1564	1564
Obfuscated Total Quality	1401203	13977089
Obfuscated Average Quality	1635	1630

Table 5.1: Results for News Article

Analysis

This piece of data provided a slightly disappointing bitrate of 0.7 bits for every 10 words. A possible explanation of this is that the text contains a large number of names and numerical data, such as times. These are generally not in the WordNet database and so will not be able to contain data, and also may cause words that are suitable for obfuscation to fail the quality checks if the follow or proceed a name or time.
There were no incorrect bits found, which is a good sign, although there were 10 cases where words were found to have bits where there were none.
A surprising result is in the quality tests. In this piece of text, the quality was not decreased, in fact there is a noticeable improvement in the quality score! This will help prevent against automatic detection. There are, however, some cases in the output where the replacement word will arise suspicion amongst humans who carefully read the text, such as "**fourth** gold medal" being replaced with "**quarter** gold medal", and "four **million**" replaced by "four **billion**".

5.4.2 USENET Postings

The USENET postings were taken from the Westbury Lab USENET corpus (2005-2010) [31]. This data was chosen as it is freely accessible and the language used in the text is from normal internet users and so is a good test for the algorithms effectiveness for "web language" text where there may be many spelling and grammar mistakes. There are three different postings totalling 616 words, with any usernames or other personal information removed. The topics are email clients and the naming of villains in fiction.

Results

	Bitstream 1	**Bitstream 2**
Words	616	616
Total Bitrate	64	63
Average bitrate (/10 words)	1.03	1.02
False Positives	11	16
False Negatives	1	0
True Positives	66	59
True Negatives	538	541
Error Count	0	0
Original Total Quality	579641	579641
Original Average Quality	940	940
Obfuscated Total Quality	684284	647641
Obfuscated Average Quality	1110	1051

Table 5.2: Results for Usenet Postings

Analysis

This text file is perhaps the most successful of all of the files processed. It has achieved a good bitrate (just over 1 bit per 10 words) and also has achieved an improvement in the quality.
The false positive rate is consistent with the other text files, and it does experience a single false negative with the first bitstream. There were no errors found in the results. Interestingly, even though the bitrate was only very slightly lower for the second bitstream, the number of false positives is considerably higher. This will be due to replacement words failing deobfuscation checks with the available bit, but the original word being deobfuscatable.

5.4.3 Academic Paper Extract

The academic paper is the introduction section from Ross Anderson's paper "Information Hiding - A Survey" [27]. This text contains reference to some technical terms and names

and consists of 964 words.

Results

	Bitstream 1	**Bitstream 2**
Words	964	964
Total Bitrate	82	74
Average bitrate (/10 words)	0.85	0.76
False Positives	18	17
False Negatives	3	2
True Positives	66	66
True Negatives	877	879
Error Count	0	0
Original Total Quality	876195	876195
Original Average Quality	908	908
Obfuscated Total Quality	927529	924523
Obfuscated Average Quality	962	959

Table 5.3: Results for Academic Paper

Analysis

The results for the academic paper are as expected for this type of text. The bitrate is lower that some of the other documents as the text contains a number of names and technical terms, which will not be in the WordNet database. They will also affect the quality tests of words surrounding them, meaning that some words will not have had data hidden even if they supported it.
The quality of the text is again slightly lower than other documents, as I expected. This is, as I have mentioned, due to the number of technical terms and names in the text which will not appear in the frequency data.
The false positive rate is again consistent with the other documents. There are a few false negatives, which is a slight problem, and these will be explained later. No errors were found.

5.4.4 Fiction Text

This piece of text is an excerpt from the first chapter of Peter Kay's autobiography "The Sound of Laughter". It was chosen as the language is not as formal as in some of the other texts [17].

Results

	Bitstream 1	**Bitstream 2**
Words	1969	1969
Total Bitrate	190	182
Average bitrate (/10 words)	0.96	0.92
False Positives	23	30
False Negatives	2	2
True Positives	167	161
True Negatives	1777	1776
Error Count	0	0
Original Total Quality	2135362	2135362
Original Average Quality	1084	1084
Obfuscated Total Quality	2253878	2256190
Obfuscated Average Quality	1144	1145

Table 5.4: Results for a Piece of Fiction

Analysis

While the false positive rate seems high, it is not more than expected for a block of text of that length. The unexpected result is the false negative rate, meaning that for two words in which data was hidden no data was found. Having run the tests again there is no obvious reason for this to occur, no bits are found in the words. The only explanation that could be a possibility is in storing the output some extra whitespace has appeared, and has not been removed by the calls to the trim() function. More investigation will be required to find this problem. This is still a very small issue as it only affects 2 words in 1900 (less than 0.1% of the words). The quality of the text (in the eyes of automatic analysis using bigram frequencies) has again improved.

5.5 Comparison of Documents

There are some differences between the four different documents. The bitrate varies between them by as much as 0.2 bits for every 10 words. The lower bitrates can be found in the news article and the academic paper extract. This is due to the number of names, technical terms and numbers in these two documents, which are not contained in the WordNet dictionary. These terms also affect the quality ratings for the surrounding words, limiting the bitrate further. The USENET postings and book extract, however, contain less of these specialised terms and so there are more words that can have bits included.
All four of the documents experienced an increase in the quality after they had been processed, due to the quality testing and quality centred synonym retrieval. What was surprising

is that the news article, despite all of its name and numbers, had the best per word average quality. This may be due to it being written by a professional journalist so there will be very few poor qualities to begin with. It is surprising that the academic paper is so much lower that this with an average per-word quality of just 962, compared to the news article's 1635. A possible explanation for this is that text contains a large amount of punctuation, and some technical terms as well as names and numbers, which affect the quality test as specialised terms are not common.

5.6 Results Discussion

As can be seen from the data above, there is a noticeable difference between the output of the algorithm when applying different bitstreams to the same text. In these tests, the second bitstream generally generated a lower bitrate than the first, and all (apart from the fiction text, which experienced a slight increase), resulted in a lower quality than when using the first bitstream.
The aim for the bitrate was to achieve a rate of around one bit for every 10 words. While, on average, the performance of the algorithm is just shy of that amount, it is very close. Apart from the false positives and the false negatives, there were no cases where the algorithm deobfuscated the wrong bits from a word.
On examining the output from the algorithms (which is included with the original text on the cd), there are cases where the outputted words are not suitable for the surrounding text. While generally the sentence still has meaning, the words will cause suspicion amongst human readers. This is tested in more detail though the user survey in section 6.7.

I will now look at some of the more interesting results in more detail:

5.6.1 False Positives

There is a known reason why false positives occur when deobfuscating. There are a number of words which will always appear to contain bits even if they do not. This is often a case with words that can only contain a 0 (but the bitstream required a 1), and so fail the deobfuscation test when obfuscating.
A possible fix for this would be to implement a blacklist of words which have this affect. These could be found by running the algorithm on a much larger corpus than the test data and logging the words where this occurs. These can then be ignored when obfuscating and deobfuscating. This may limit the bitrate slightly but will improve the correctness of the algorithm.

5.6.2 Improved Quality

Perhaps the most surprising result of the tests is the improvement in the quality value of the texts using the bigram frequency data. The reason that this has occurred is due to the

way in which the synonym sets are built. The lists are ordered according to the frequency of each possible word and the word before (in the text). This means that whenever a word is substituted, the word will be one of the top three in this list (depending on which bit is hidden). Quite often, these will be of higher quality that the original word. This, combined with the test performed during obfuscation (which does not hide bits in words if the resulting quality is too low) is the reason that the quality generally increases.

5.6.3 False Negatives

After more inspection of the output of the program, this appears to be due to some bug in the implementation. The example from the usenet output is the word "short", preceded by "a". The program is not able to generate a synset for this word when deobfuscating. The synsets are generated in exactly the same way when deobfuscating as when obfuscating, and the words where this issue occurs all generate synsets while obfuscating. This is a low priority issue as the program and algorithm are only a prototype and this issue appears to be related to the implementation rather than the design of the algorithm. It is also worth noting that Chang and Clark [4] experience a small number of false negatives also.

5.7 User Survey

To test the robustness to human analysis aim of the algorithm a short survey was prepared which contains output sentences from the documents used for the statistical testing. The subject is presented with 10 sentences, split into two parts. For the first part they do not know anything about the algorithm, and for the second part they receive a brief explanation. The subject is asked to first give the sentence a brief glance and assign the sentence a score of 1-10, where 1 means that the sentence seems perfectly normal and 10 means the sentence does not make any sense at all or seems completely wrong. They are then asked to study the sentence in greater detail, and provide the score again. They are then free to make any comments on the sentence, and are encouraged to circle any words that they feel are incorrect or out of place.
As a control, in each section one of the sentences (number 3) are the original text without the algorithm applied and no data hidden. This is control data to test if the algorithm does truly impact the text. There is at least one sentence in each section from each of the four documents processed.
The survey is visible as Appendix A.

5.7.1 Results

The survey was completed by six people. Two of these were non-English native speakers. The average score was taken for each sentence and is presented in the table below. It is

worth noting that a lower score is preferable.

Question	**Glance Average**	**Examine Average**
Part 1, 1	7	5.6
Part 1, 2	4	3.66
Part 1, 3 (Control)	2.83	3
Part 1, 4	5.66	6.5
Part 1, 5	3	2.5
Part 2, 1	2.6	2.3
Part 2, 2	1.3	1
Part 2, 3 (Control)	4.8	7
Part 2, 4	2.1	4.1
Part 2, 5	2.3	2.6

Table 5.5: Results for the User Survey

The results are generally as expected. The subjects in most gave the sentence a lower score when they read them more carefully than when they just glanced at them.
Firstly, for the two control questions (1,3 and 2,3), the users found that the first sentence did not have much of a problem, except for a lack of commas. The second sentence, however, did cause them problems. The general comments were "What does dupes mean?", and that the word fertile seemed out of place in the sentence. This was expected for the word "dupes" as it is not a very common word in the English language; while the "fertile", even though it is perfectly acceptable to use in this context, is not often used.
The two news article sentences (1,2 and 2,4) were both well received. Sentence 1,2 was at first given a slightly lower score, but the users improved their scores when reading the sentence closer. This is due to the fact that none of the words are particularly out of place (although the choice of wording and lack of commas was noted). Sentence 2,4 contained the "quarter gold medals" phrase; this was expected to have a large negative effect on the score but on the first glance it did not (only on the closer look did the subjects revise their scores).
The two academic paper sentences (1,5 and 2,5) were both scored well. The words that were replaced in these sentences, such as "some" with "most", without previous knowledge of the subject topic still make sense. The only comments about these sentences were that the subjects did not know what some of the words meant.
The USENET sentences (1,4 and 2,2) produced some interesting results. 2,2 received a perfect score. This is because the sentence was already well formed, and the data is hidden in the word "example", which is not changed from the original. The sentence 1,4 was not as well received and received one of the worst scores. This, from the comments, was due to the phrases "completely I use it for" (where "completely" replaces "all"), and "to dress my

memory " (where "dress" replaces "jog"). These are two bad replacements, hence the low score.
Finally, the two sentences from the piece of fiction (1,1 and 2,1) provided some interesting results. The first sentence, 1,1, received a very low score despite the only word difference from the original being "door" replaced by "brink". The general consensus among the subjects that the sentence did not make sense. This may be due to the lack of context. If the subjects had access to the surrounding text the sentence would make more sense, but taken by itself it does not. The second sentence, however, received a much better score, as the only replacement was the word "top" with "point". This still made sense to the subjects, although two of them did comment that the word "point" may contain data. One thing that was found during these tests is that the non-native English speakers had more trouble identifying suspicious words than the native speakers. This means that the algorithm will be more effective if used on English text in non-English speaking countries. This is a similar result as described by Shirali-Shahreza et al. [32] with their American-British English substitutions.
It was also interesting that the scores were generally better once the subjects know about the algorithm and how it operated. Although the sentences may not be quite as complex as some of the sentences in the first section, it is possible that knowing what to look for made the users more tolerant to bad words, at least on a subconscious level.
What this test has proven is that the quality of the original text can have a large impact on the human analyser, even simple punctuation mistakes (all punctuation is kept by the algorithm) caused them to lower their scores in some places. Words which the users did not know, such as "dupes", or uncommon usages of words threw off the subjects, which will cause human analysis to throw up false positives.

5.8 Robustness to Steganalysis

5.8.1 Automatic

As has been shown with the results from the statistical analysis above, if the output was to be analysed by a method which used frequency analysis to detect hidden data then it should not be detected. It could, however, be susceptible to more advanced steganalysis techniques, such as the use of a grammar or more advanced semantic testing. There is a good possibility, however, that the algorithm will pass the simple frequency analysis and not be suspicious enough to warrant the more advanced methods, which will have a much higher cost.
There is also the fact that with this form of steganography, there is no easy way to know if the text actually contains data or not. If the deobfuscation algorithm is applied to unprocessed text, then there is a good chance that it will come out with data anyway, as the

deobfuscation algorithm assumes that data is hidden. So if an attacker were to run the algorithm on any piece of text he finds, it will always return results. This ties into the chosen stego attack, as the attacker have access to both the algorithm and the stegotext but would not be able to tell if the data he was extracting was meaningful or not.

5.8.2 Human

Taking into account the user survey results and studying the output from the algorithm, the algorithm does provide a good level of protection from human analysis. In general, the words that are used for replacements fit in with the surrounding sentence. One thing that causes more problems to human analysers is errors in the original text, meaning that the strength of the algorithm is related to the quality of the original text.
There are cases where the word that is used as a replacement does not make sense. As this is a prototype algorithm, this was expected, and will require further work to improve (although it will be extremely difficult to remove all cases).
The implementation of the "bad words" list to remove words from the system which cause false positives to be found when deobfuscating is another improvement that can be made to the algorithm. This could be done either through user reporting on the live application or by using the test application and much larger sets of test data that was used in the evaluation.
The algorithm will be stronger when used on English text in countries where English is not the native language. Even the test subject who has been speaking English for four years almost exclusively had trouble understanding some of the words used; not that they didn't make sense in the sentence but they didn't recognise the words themselves.

Chapter 6

Conclusion

6.1 Future Work

There are a number of improvements that can be made to the algorithm in the future. The primary element that could be improved the the synonym retrieval method. This is the primary reason why some words that are returned are not in keeping with the text. WordNet provides some more links between words, such as word1 “is a” word2, which could be used. One thing that was disabled in WordNet for this programs are exception rules, which show replacements for words that are not in the database, such as “completed” goes to “complete”. This were removed as they did not fit the synonym retrieval algorithm, but if they could be worked n they would help improve the bitrate.
The quality test could be improved by comparing words with the entire body of text, rather than just the preceding word. This will help keep words in context with the entire text.
The StegChat application, while only a low level prototype, could be expanded into a full fledged product. It would require a large number of extra features, such as being able to check the online status of contacts. A more substantial desktop application could also be produced, using the same principle as the evaluation program.

6.2 Conclusion

The goal of this project was to produce an lightweight and robust text steganography algorithm using the idea of synonym substitution. The output should be robust against both automatic and human analysis, simple enough for any user to understand, and lightweight enough that it can be performed on low power machines, and even by hand.
The algorithm in itself is very lightweight. You could give someone a printed version of the dictionary and corpus data, and with little explanation they would be able to perform the algorithm by hand (albeit with no speed!).

The output can be printed, copied, retyped, sent in emails but the data will never be lost (save for spelling mistakes while copying). The output cannot be compressed, so it is not vulnerable to compression algorithms which is one of the major flaws in many image, audio and video steganography algorithms.
The desired capacity for hiding data was around one bit for every ten words. The actual capacity, found through the statistical evaluation, is around 0.9 bits for every 10 words, so this has very almost been achieved.
The output of the algorithm is robust against at least frequency analysis based methods of steganalysis, and in many cases would be robust against more advanced methods involving grammars. Through human evaluation, the algorithm has also been moderately successful in avoiding human detection. There is still work that needs to be done, as described in the future work section above, but for a prototype the algorithm is very successful. Through the development of the test application, StegChat, the algorithm has proven that it can work in a real life situation and with further development the StegChat application could become a very useful tool.

Bibliography

[1] Krista Bennett. *Linguistic Steganography: Survey, Analysis, and Robustness Concerns for Hiding Iformation in Text*. West Lafayette, Indiana, USA: Centre for Education, research om Onformation Assurance, and Security, Purdue University, 2004.

[2] Igor Bolshakov. "A Method of Linguistic Steganography Based on Collocationally-Verified Synonymy". In: *Information Hiding*. Ed. by Jessica Fridrich. Vol. 3200. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pp. 607–614. URL: http://dx.doi.org/10.1007/978-3-540-30114-1_13.

[3] *British National Corpus*. 2010. URL: http://www.natcorp.ox.ac.uk/.

[4] Chin-Yun Chang and Stephen Clark. "Practical Linguistic Steganography using Contextual Synonym Substitution and Vertex Colour Coding". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Ed. by Practical Linguistic Steganography using Contextual Synonym Substitution and Vertex Colour Coding. MIT, Massachusetts, USA, Oct. 2010, pp. 1192–1203.

[5] Jeff England. *Audio Steganography. Echo Data Hiding*. URL: http://www.ee.columbia.edu/~ywang/MSS/Project/Jeff_England_Audio_Steganography.ppt.

[6] Alex Franz and Thorsten Brants. *Google Web IT n-gram corpus*. 2006. URL: http://googleresearch.blogspot.co.uk/2006/08/all-our-n-gram-are-belong-to-you.html.

[7] Jessica Fridrich, Miroslav Goljan, and Dorin Hogea. *New methodology for breaking steganographic techniques for JPEGs*.

[8] Sean Gallagher. *Steganography: how al-Qaeda hid secret documents in a porn video*. arstechnica. May 2012. URL: http://arstechnica.com/business/2012/05/steganography-how-al-qaeda-hid-secret-documents-in-a-porn-video/.

[9] Sharon Gaudin. *Record Broken: 82% of U.S. Email is Spam*. 2004. URL: http://itmanagement.earthweb.com/secu/article.php/3349921.

[10] Google. *Google App Engine Developer Pages*. Google Inc. 2012. URL: https://developers.google.com/appengine/.

[11] American Prosecutors Research Institute, ed. *Steganography: Implications for the Prosecutor and Computer Forensics Examiner*. Child Sexual Exploitation Program Update 1.1 (2004). URL: http://www.ndaa.org/pdf/Update_gr_v1_no1.pdf.

[12] Neil Johnson. "Steganalysis: The investigation of Hidden Information". In: *IEEE Information Technology Conference*. Ney York, USA.

[13] Neil Johnson and Sushil Jajodia. "Steganalysis: The Investigation of Hidden Information". In: *Proceedings of the 1998 IEEE Information Technology Conference*. New York, USA, 1998.

[14] James C. Judge. *Steganography: Past, Present, Future*. 2009. URL: http://www.sans.org/reading_room/whitepapers/stenganography/steganography-past-present-future_552.

[15] United States Department of Justice. *Criminal complaint by Special Agent Ricci against alleged Russian agents*. 2010. URL: http://www.justice.gov/opa/documents/062810complaint2.pdf.

[16] D. Kahn. *The Codebreakers - The Story of Secret Writing*. New York, New York, U.S.A: Scribner, 1996. ISBN: 0-684-83130-9.

[17] Peter Kay. *The Sound of Laughter*. Century, 2006. Chap. 1. ISBN: 978-1846051616. URL: http://www.randomhousesites.co.uk/catalog/extract.htm?command=search&db=main.txt&eqisbndata=009950555X.

[18] Adam Kilgarriff. *BNC database and word frequency lists*. 1996. URL: http://www.kilgarriff.co.uk/bnc-readme.html.

[19] G. Kipper. *Investigator's Guide to Steganography*. 2003. URL: http://flylib.com/books/en/1.496.1.8/1/.

[20] George A. Miller. *About Wordnet*. Princeton University. 2012. URL: http://wordnet.princeton.edu/.

[21] T. Moerland. *Steganography and Steganalysis*. Leiden Institute of Advanced Computing Science. 2003.

[22] Tayana Morkel, Jan H P Eloff, and Martin S Olivier. "An Overview of Image Steganography". In: *Proceedings of the Fifth Annual Information Security South Africa Conference ISSA2005)*. Ed. by Les Labuschagne Hein S Venter Jan H P Eloff and Mariki M Eloff. Published electronically. Sandton, South Africa, June 2005.

[23] Shishir Nagaraja et al. "Stegobot: a covert social network botnet". In: Information Hiding 2011. 2011. URL: http://www.cs.bham.ac.uk/~nagarajs/papers/stegobot.pdf.

[24] BBC News. *2012 Paralympics: Huge crowds cheer David Weir to victory*. BBC. Sept. 2012. URL: http://www.bbc.co.uk/news/uk-19535236.

[25] Cosimo Oliboni. *OpenPuff*. URL: http://embeddedsw.net/OpenPuff_Steganography_Home.html.

[26] *OpenStego*. URL: http://openstego.sourceforge.net/.

[27] Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. "Information Hiding - A Survey". In: *Proceedings of the IEEE, special issue on protection of multimedia content*. Vol. 87. July 1997, pp. 1062–1078.

[28] American National Corpus Project. *ANC First Release Frequency Data*. 2010. URL: http://www.americannationalcorpus.org/frequency.html.

[29] Pierre Richer. *Steganalysis: Detecting hidden information with computer forensic analysis*. 2003. URL: http://www.sans.org/reading_room/whitepapers/stenganography/steganalysis-detecting-hidden-information-computer-forensic-analysis_1014.

[30] Seth Schoen. *Secret Code in Color Printers Lets Government Track You*. Electronic Frontier Foundation. 2005. URL: https://www.eff.org/press/archives/2005/10/16.

[31] C. Shaoul and C. Westbury. *A USENET corpus (2005-2010)*. University of Alberta. 2011. URL: http://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html.

[32] M.H. Shirali-Shahreza and M. Shirali-Shahreza. "A New Synonym Text Steganography". In: *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IIHMSP '08 International Conference on*. Aug. 2008, pp. 1524 –1526. DOI: 10.1109/IIH-MSP.2008.6.

[33] M.H. Shirali-Shahreza and M. Shirali-Shahreza. "Text Steganography in chat". In: *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*. Sept. 2007, pp. 1 –5. DOI: 10.1109/CANET.2007.4401716.

[34] Brett Spell. *Java API for WordNet Searching (JAWS)*. Computer Science and Engineering (CSE) department at Southern Methodist University. 2008. URL: http://lyle.smu.edu/~tspell/jaws/index.html.

[35] Umut Topkara, Mercan Topkara, and Mikhail J. Atallah. "The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions". In: *MM&Sec '06: Proceeding of the 8th workshop on Multimedia and security*. Geneva, Switzerland: ACM Press, 2006, pp. 164–174. ISBN: 1-59593-493-6. DOI: http://doi.acm.org/10.1145/1161366.1161397.

[36] Johannes Trithemius. *Steganographia*. Frankfurt, 1499.

[37] John Walker. *Stego!* 2005. URL: http://www.fourmilab.ch/javascrypt/stego.html.

[38] Peter Wayner. *spammimic*. 2000. URL: http://www.spammimic.com.

[39] Wikipedia. *Image containing hidden message*. 2009. URL: http://en.wikipedia.org/wiki/File:Avatar_for_terrorist.png.

Appendix A

User Survey

Questionairre

Please answer these questions without looking at second page. For each sentence, please give a score out of 10, with 10 being suspicious and 1 being absolutely perfectly formed. Please score the sentence in the first box after a quick read, and the second box after a closer inspection. An error can be a word that does not fit the sentence, an incorrect spelling etc. Then, make any note about the sentence describing what you think is wrong or doesn't make sense. (Also feel free to circle any words that are particularly noticeable)

1. I mean them stood at the brink in green tights and holding a scroll like those blokes out of Shrek 2.
   Glance Score: ☐ Examine Score: ☐
   Comments:

2. I swam the best I could and to get off with personal bests in every case is brilliant.
   Glance Score: ☐ Examine Score: ☐
   Comments:

3. I've got the firewall and dual virus shields, but if they do make it through I sure don't want to give them my (and all my family and friend's) email addresses!
   Glance Score: ☐ Examine Score: ☐
   Comments:

4. And I will surely have a look at Thunderbird, but in truth, what I find useful about Outlook and completely I use it for (as I don't use it to email) is to dress my memory to pay bills, and do this and do that.
   Glance Score: ☐ Examine Score: ☐
   Comments:

5. Recent attempts by most governments to limit online free speech and the civilian use of cryptography have spurred people concerned about liberties to make techniques for anonymous communications on the net, including anonymous remailers and web proxies.
   Glance Score: ☐ Examine Score: ☐
   Comments:

Part 2

The algorithm that has been developed uses synonym substitution to hide data in text. Words are replaced with their synonyms (words which have the same meaning) by choosing the first synonym to hide a 0, the second to hide a 1 and the third to hide 1,0. The data can then be retrieved by the receiver. Knowing this information, please evaluate the following sentences in the same way as before.

1. Just push the button in at the point with your thumb and lower it slowly.
   Glance Score: ☐ Examine Score: ☐
   Comments:

2. For example there is a Jim Moriarty who is an attorney here in Houston.
   Glance Score: ☐ Examine Score: ☐
   Comments:

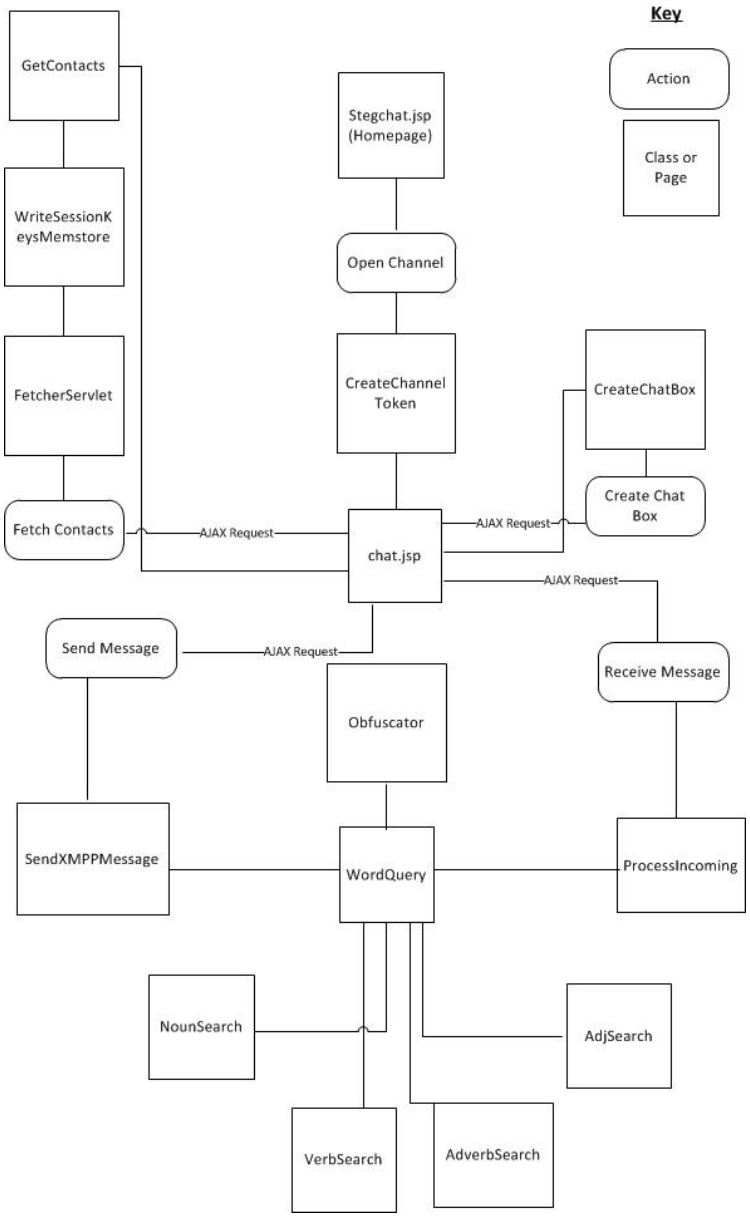
3. I find the phone book a fertile source of names but hesitate to use exact dupes of real persons.
   Glance Score: ☐ Examine Score: ☐
   Comments:

4. Thousands of spectators have lined the way of the wheelchair marathon through central London to cheer GB's David Weir to a quarter gold medal at the Games.
   Glance Score: ☐ Examine Score: ☐
   Comments:

5. Then, we will describe a number of attacks against these techniques; and finally, we will try to get general definitions and principles.
   Glance Score: ☐ Examine Score: ☐
   Comments:

Appendix B

Structure Diagram



Appendix C

CD Contents

The structure of the cd is as follows:
Report.pdf - This report in pdf format
StegChat/ - The StegChat full web application (eclipse project format)
StegChatEval/ - The test application (eclipse project format)
TestData/ - Test Data used by the evaluation program for the Evaluation results
dict/ - The wordnet dictionary files, required by the evaluation program
jaws-bin.jar - Java library required by the evaluation program

There are a number of files within the test data class. For each of the four document types, there is a data and an output file. The data files are the files which can be entered into the test program. The output file contains the two outputs of the algorithm run on the document with each of the two test bitstreams.
There are two bitstream files, testbits1 and testbits2.
There are also two csv files required by the test application to function, "bigram.csv" which contains the ANC bigram data, and "demog.csv" which contains the word frequency data from the BNC.

Appendix D

Program Run Instructions

To run the StegChat web application requires a Google account, with stored contacts. If you do not have an Google account, it is recommended that you run the local test program.

D.1 StegChat

To access the StegChat application, in you web browser go to http://stegchat.appspot.com
A Google account, with at least one saved contact is required to access the application. To communicate with another user they must be in your contacts list and also have a Google account.
Once you have signed in, click on "Authorise Contacts", you will be presented with a link to authorise contacts access. You will then be guided to the chat screen and click on "Fetch Contact"to get your contact list. You can then select a contact from the dropdown box and click create chat to open a chat connection to that user. This second user needs to perform the same task with your account to enable chatting.
Bits can be added using the "Add Bits" form, by typing the data in the form "101101...". Messages can be entered in the box below. NOTE: Long message may overun request time limits and not be processed. if you wish to test the algorithm with longer text strings use the evaluation program.

D.2 StegChatEval

The source files of this program are included. The simplest way to run the program is to load the project into eclipse, add the jaws-bin.jar library (included on the cd), and add the VM option
-Dwordnet.database.dir=C:\Users\Joe\Desktop\dict
Where the address points to the location of the "dict" folder containing the wordnet files.

This is included on the cd and should be copied onto your computer.
Running the BitstreamGenerator class will generate a txt file containing a random bitstring of 3000 length. If you wish to manually declare a bitstream, produce a text file with the bits separated by spaces e.g. "1 0 1 1 0 ..." Once the program has been set up, it can be run using the "Evaluator" class. You will first be asked to select a bit file. There are two example files in the TestData folder, testbits1 and testbits2, but any bits file made as described above can be used. You will then be asked to choose the bigram file. This is the "bigram.csv" file found in the test data folder. The program will then ask for a frequency file, this is the "demog.csv" file, also found in the TestData folder.
You will then be asked to provide an input file. This can be any of the four data files in the TestData folder, or a txt file of your making containing the text you wish to use as a covertext. Once the file is selected the program will be run and the output displayed. Once it is finished you will be asked to slect another file, this will repeat until you manually stop the program.

D.3 Code Reuse

In line with the guidelines on referencing resused code, the following three classes from the StegChat program all reuse code.
- FetcherServlet.java (entire class except for some edited variables)
- GetContacts
- WriteSessionkeysMemstore
For the 2nd and third classes, only part of the code if from the external source. The sections that are reused are clearly marked with comments.
The source of the code is:
https://developers.google.com/appengine/articles/java/retrieving_ gdata_ feeds