

A Considered Approach to HTML Steganography by James Norman

James P. Norman

976690

Supervised by Phillip James

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Master of Science (MSc)

Master of Science



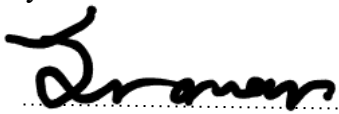
Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

September 30th, 2022

Declaration


This work has not been previously accepted in substance for any degree and is not being con- currently submitted in candidature for any degree.

Signed  (candidate)

Date **30/09/2022**

Statement 1


This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed  (candidate)

Date **30/09/2022**

Statement 2

I hereby give my consent for my thesis, if accepted, to be made available for photocopying and inter-library loan, and for the title and summary to be made available to outside organisations.

Signed  (candidate)

Date **30/09/2022**

Abstract

There has forever been a need to conceal information to a specific audience, whether this is setting up a magic trick to produce the '*impossible*' or hide a message from an enemy General. The key to effective concealment is to make the observer believe that nothing is amiss or redirect their attention to a strange but irrelevant detail. In this steganographic scheme HTML tags and code will be used to mask the concealment 'cover text'

"The purpose of steganography is to hide the very presence of communication by embedding messages into innocuous-looking cover objects, such as digital images. To accommodate a secret message, the original image, also called the cover image, is slightly modified by the embedding algorithm to obtain the stego image"[1]

Using **python-3.8** and **PyQT5** to implement a modern, carefully considered, and concealable steganographic algorithm it is possible to conceal messages within HTML files which can remain readable when opened by a webpage but also mask a hidden plaintext when viewed in a '.txt/.html' format. The algorithm and resulting program that this thesis covers takes onboard not only the factors that make a message concealable but efficient and complexity conscious to produce a HTML based steganography technique.

By incorporating this algorithm, it is possible for a user or company to watermark their work without using obscuring (image across document) or invasive (Digital rights management) and leave other details and messages that can be backread to them or another party while avoiding alerting a party which may want to alter or copy the document. This algorithm is designed to build upon existing approaches as shown by (Garg M.G, Odeh A., Elleithy K., Faezipour M., Abdelfattah E, Shahreza M.S)[2,3,4]. and similar steganography techniques used within '*Digital Text Steganography*' specifically letter encoding when applied to readable file formats. Similarly, to Garg M. G's approach this can be done by applying various table transformations and encoding symbols into the text[2], working off of this principle we can further develop less obvious and invasive steganography techniques.

Acknowledgements

I would like to take this section to thank those who have supported and encouraged me across the duration of both my three-year (BSc) and my final year as an (MSc) student. Ben Nettleship my close friend, my family, and support staff/lectures especially my supervisor Phillip James in the department of Science and Engineering at Swansea University. The support shown has been invaluable.

Table of Contents

Declaration	3
Abstract	5
Acknowledgements	7
Table of Contents	9
Chapter 1 Introduction	12
1.1 Motivations	12
1.1.1 Objective	14
1.2 Overview	15
1.3 Contributions	16
1.4 Literature review	17
1.5 Research base	36
1.6 Tools, Environment, and resources	38
Chapter 2 Algorithm and System development	44
2.1 Methodology	45
2.2 Program Structure	48
2.3 Algorithm Design	51
2.4 Findings and Evaluation	54
Chapter 3 Analysis	59
3.1 Efficiency and Effectiveness	59
3.1.1 Program Efficiency	60
3.1.2 Program Efficacy	64
3.1.3 Program Interface	65
3.2 Comparative Analysis	67
3.3 Discussion	70
3.4 Findings	71
3.5 Summary	72
Chapter 4 Conclusions and Future Work	74
4.1 Conclusion	74
4.2 Future Work	75
Bibliography	77
Appendix A Time Cost Gantt Chart	84
Appendix B Writer and Reader Specific Algorithms	86

Chapter 1

Introduction

Not to be confused with cryptography ("*the practice and study of techniques for secure communication in the presence of adversarial behaviour.*") [5] steganography does not seek to make communication itself 'secure' but to reduce the footprint of communication to particular parties this is done through a masking process with a cover-text, image, and other various forms of media to leave adversaries unaware of a steganographic encoded file. The earliest mention of cryptography being employed and researched comes from the 'Cryptographia'

(*Cryptographia: oder Geheime schrift- münd- und würckliche Correspondentz*) [8]

(*Cryptographia: or Secret Writings and actual Correspondents*) with many examples of cryptographic schemes with much overlap showing origin into what we would consider steganography.



(Fig 1.1) Johannes Balthasar Friderici. *Cryptographia, oder, geheime schrift*. Hamburg, 1685.[7]

How can we take advantage of this fact to create a unique algorithm for HTML based steganography without raising suspicion and keeping functionality of the language itself?

This project is aimed at the topic of digital text steganography, the practice and study of techniques for masking text data within a seemingly legitimate file. This can apply to a multitude of formats but for the focus of the project this will only involve '.html' and resulting '.txt' files to store the data in a raw format. Digital text steganography originates from basic physical text steganography before messages were more

commonly passed digitally, these schemes often overly simplistic would be physical like encoding a message on sheet music[6] or acrostic. Now that the field is firmly in the digital age it is somewhat undermined by encryption despite the ability for both to be used together, however a few modern use cases remain relevant:
(Watermarking, identification, puzzles/challenges)

1.1 Motivations

Research question

To condense the aims and objectives of this project initially into a research question; Can an algorithm that encodes a HTML file be created that fulfils three requirements, has a low level of detection, maintains the structure and integrity of the cover text, and successfully encodes a variety of complex messages. These three requirements detection to ensure

secrecy (human observable), integrity to preserve cover text (visible difference), and compatibility to allow a broader variety of messages (program observable) create an effective steganography scheme, and as a result the project can be judged in accordance with these criteria. This research question can be partly answered by other papers[2,3,4] which tackle the same field but encode in a manner where the cover-text is manipulated to a noticeable degree or cases where the cover text is produced by the encoding itself. This goes against the requirement of integrity; the project algorithm will be built to maintain this on top of current research in digital text steganography.

Producing an algorithm that fulfils these three criteria will provide a more effective standard for the current modern applications of steganography, in example this could involve a company or individual encoding signatures onto code or documents that they produce.

The first requirement will ensure that the encoding is not noticeable when the document is read, this will improve clarity of the document on top of secrecy.

The second requirement concerning integrity of the algorithm will see that the document retains its original format, this tenant can be important when the cover text is still critical information like in the case of a watermarked document. In use the third requirement of compatibility allows the company or individual to encode a wide variety of messages, this can even include encrypted text as it may be beneficial to avoid its hallmarks to avoid arousing suspicion.

1.1.1 Objectives

As stated in the project 'Initial report' the three objectives were:

1. *'To create a suitable, efficient, and informative application to produce examples of digital text steganography'*
2. *'To enable and automate the creation of simple HTML based steganography techniques'*
3. *'To evaluate the efficiency of the developed algorithm and compare/contrast other related HTML based approaches'*

These initial objectives can be developed to be further tailored to the project, requirements, and research question. For this project, the main objective is to create an algorithm that encodes a HTML file that fulfils three requirements, has a low level of detection, maintains the structure and integrity of the cover text, and successfully encodes a variety of complex messages. These requirements as part of the first objective cover the algorithm but it is also important to consider objective two alongside the first. This will provide a user interface to aid in automation and better suite the algorithm to be used by a company or user more in line and realistic to the potential use cases of this project.

After revision the objectives have been divided into more tailored and appropriate goals:

As a main objective (primary):

Develop an algorithm which encodes a plaintext message within the .html format

objective milestone 1.1 •This algorithm should have a low level of detection and be hard to spot to the average user

objective milestone 1.2 •This algorithm should maintain the structure of the original cover text without distorting function or meaning

objective milestone 1.3 •This algorithm should be capable of encoding a variety of messages without breaching milestone requirements 1.1 and 1.2

Revised objective one:

1. To develop an informative, efficient, and intuitive user interface that can produce encoded and decoded steganography text using the developed algorithm

Revised objective two:

2. To enable and automate the process of saving and loading large plaintext and steganography encoded files

Revised objective three:

3. To evaluate the efficiency and effectiveness of the developed algorithm and compare/contrast other related HTML based approaches

1.2 Overview

The remainder of '*Chapter 1: Introduction*' sections *1.3 Contributions*, *1.4 Literature Review*, and *1.5 Tools, Environment, and resources* describes the research, reading, and setup of the project to help better justify and understand design decisions and research which have affected the direction and motivation behind development.

Review of contributions and relevant material of this project will also be referenced and quantified within '*Chapter 3: Critical Analysis*'

'*Chapter 2: Algorithm and System Development*' will discuss the development timeline along with decisions and background responsible for the outcome of the algorithm, also covering important findings and evaluating the final design in relation to the requirements, aims, and objectives. This will be then later brought into context with other approaches from '*Chapter 3: Critical Analysis*' where pros cons and other efficiency factors will be argued in context of the research question

In '*Chapter 3: Critical Analysis*' the chosen algorithm will be closely compared and evaluated statistically in comparison to other available approaches based upon the requirements of an effective steganography algorithm and the potential uses of each when applied in a real-life scenario. This will help better place each algorithm in a context for evaluation and conclude thoughts in '*Chapter 4: Conclusion and Future Work*'.

1.3 Contributions

The main contributions of this thesis and program/algorithm can be seen as follows:

- **A Secretive, Effective, and Compatible HTML Steganographic algorithm**

This algorithm fulfils each objective and is fully documented and commented for clarity. Fully compatible to be integrated into a user interface for simplicity and repeated processing of new text.

- **A Clear, Efficient and Compact user interface**

The interface is fully integrated with the algorithm and aids a variety of users by clearly presenting each function of the algorithm, with some quality-of-life enhancements to aid usability and instructional guide.

- **A Review, Comparison, and Analysis of related algorithms in Digital Text Steganography and HTML Steganography**

Each comparable approach evaluated and quantified regarding the objectives and each other with benefits and drawbacks accurately labelled for each, quantifiable drawn into this comparison to better gauge efficiency.

- **An evaluation of the chosen algorithm in relation to other algorithms and performance within the objectives overall**

The comparison from the previous contribution is used to evaluate the chosen algorithm, performance and operation are first considered to better place this algorithm in comparison to others with reasoning and thorough analysis of statistical data and objectives met.

1.4 Literature Review [2,3,4,8] - [29]

Introductory Studies

Cryptographia F.J Balthasar & The history of steganography D. Khan

[8][9]

The early concepts of steganography came from a need for concealment as is evident from early graphs and Figures from

Friderici, Johann Balthasar's 'Cryptographia' from 1684[8]. Many early concepts of puzzles and secrets which would later evolve into cryptography, earlier and less secure ideas (transposition, displacement, visual, symbolic p.16,39,196 [8]) would birth the origin of steganography. *D. Khan* makes the observation that secrecy is an embedded behaviour with a clear distinction in the advancements a human may make to better conceal in a modern setting

"The origin of steganography, it seems to me, is biological or physiological.

Examples of what we call steganography today abound in the animal kingdom.

When dogs attempt to do something secret and are caught, they sometimes look a little ashamed."[9]

It is important to know historical background when considering techniques embedded within the visual and mental cues of humans

early needs and attempts to create physical text steganography highlights an evolution of techniques before the digital age.

D. Khan provides early and late examples of techniques to give a broader overview of the foundations of steganography. *D. Khan* uses an incredibly early example from 6th century BC

"A man named Harpagus killed a hare and hid a message inside its body. He sent it with a messenger who pretended to be a hunter."

This example is one of the earliest documented instances of steganography and displays thoughts behind physical concealment which would ultimately lead to the development of physical text-based techniques such as sympathetic inks. Although these findings are not directly related to digital text steganography *D. Khan's* background information helped to better focus the project definition by deciphering all meaning behind the word Steganography.

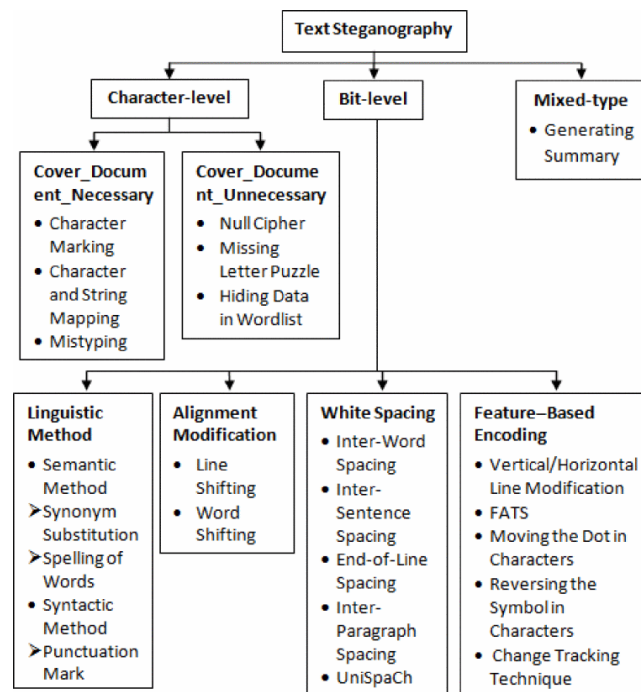
Ultimately the true origin of steganography will be unknown within humans as *D. Khan* mentioned could stem from so many '*biological*' and '*physiological*' it is hard to pinpoint. Regardless these ideas have stayed and formed the predecessor to digital techniques.

An overview of text steganography R.K, P.T, M.B

[10]

Steganography has become a diverse and evolved field that the numerous techniques available often share some overlap and ambiguity, it is not only important to distinguish formats (physical, digital, psychological) of techniques from each other but also, the precise cover that they use. Text steganography is the largest technique group and has many different approaches to concealment,

R.K, P.T, and M. B's 'An overview of text steganography'[10] helps to elucidate this, focusing on Text steganography, which is directly related to the project, then seeking to further break down text steganography into diffuse fields providing information on techniques which have been researched and proven.



(Fig 1.2) R. Bala Krishnan, Prasanth Kumar Thandra, M. Sai Baba. An overview of text steganography,2017.[10]

Referencing the different digital steganography fields (Text, Image, Audio, Visual, Network) and the origins of steganography in Greece briefly in 'SECTION I.' a fact that is corroborated among other sources[9] this introductory section serves as a lead into the classification of text-based techniques relevant to this project.

This source uses a Figure to act as a reference for structuring the rest of the paper and for this reason the information within the Figure (see Fig 1.2.) give a visual overview of techniques sub-dividing into 'character' and 'bit-level' steganography which is apparent through language and letter observation and techniques related to a machine level of processing these characters, respectively. Also covering the psychological side with 'Cover_Document_Unnecessary' which blankets techniques related to a secret which is not shared within the document itself

"Character-level embedding technique directly embeds the secrets as characters inside the text document. Depending on the necessity of the cover work/document, this type of embedding can further be classified into two categories as

Cover_Document_Necessary and Cover_Document_Unnecessary"[10]

This project uses 'Character-level Embedding' more specifically 'Character Marking'(sect. 1.1.1) a technique described in this source as:

"As a way of embedding the secrets, the respective characters in the cover document can be marked serially."[10] Other notable technique fields that are mentioned in this source and considered for this project are: End of Line embedding, Missing Letter Puzzle, Punctuation Mark; However, these would compromise the objective requirements of the system. The paper goes on to describe the 'bit-level' techniques, these techniques dealing with low level encoding and the way the text information is understood by the computer and changing the source of the cover were a temperamental approach to encoding a HTML document as functionality needed to be maintained to improve secrecy. Moving the Dot in Characters, Reversing the Symbol in Characters, and Word Shifting are used as examples of the 'bit-level embedding' approaches were considered but not chosen as they would interfere with the readability and integrity of the HTML document (see sect. 2.3 Algorithm Design)

Technique	Type of Embedding	Embedding Capacity (Approximate)
Character Marking [4] [7] [8] [9] [10], Mistyping [7] [8] [11] [12]	Character-level	Variable (due to the non-uniform occurrence)
Character and String Mapping [4]	Character-level	2-bits/cover-character
Missing Letter Puzzle [3]	Character-level	8-bits/10.5-cc
Hiding Data in Wordlist [3]	Character-level	8-bits/10.5-cc
Synonym Substitution [15]	Bit-level	1-bit/4.5-cc (best case)
Spelling of Words [16]	Bit-level	1-bit/4.5-cc (best case)
Line Shifting [7] [18]	Bit-level	1-bit/180-cc
Word Shifting [7] [18]	Bit-level	1-bit/15.5-cc
Inter-Word Spacing [19] [20] [21]	Bit-level	1-bit/10-cc
Inter-Sentence Spacing [21]	Bit-level	1-bit/166-cc
End-of-Line Spacing [21]	Bit-level (2-bits)	2-bits/60-cc
UniSpaCh [14]	Bit-level (2-bits)	1.046-bits/cover-character
Vertical/Horizontal Line Modification [23]	Bit-level	Variable (due to the non-uniform occurrence)
Change Tracking Technique [24]	Bit-level	0.33-bits/4.5-cc
Generating Summary [25]	Mixed-type (2-bits)	2-bits/82.5-cc

where, cc – cover-characters

(Fig 1.3) R. Bala Krishnan, Prasanth Kumar Thandra, M. Sai Baba. Space comparison of various techniques,2017.[10]

This source continues to evaluate and conclude that

“Although media types other than text can be used as a cover medium, organizations might prefer text documents as they are widely used in such environment.”[10]

This is a fair evaluation but lacks the detail included in the evaluation which could be brought in to support this. The paper breaks down each listed technique outlining a rough efficiency based upon the characters that can be encoded per average number of characters. The paper lists ‘UniSpaCh’

(A method by: Lip Yee. Por, Wong KosSheik, and Kok Onn. Chee; UniSpaCh: a text-based data hiding method using Unicode space characters[32]. Relying upon creating more whitespace to encode between symbols, words, and lines)

and ‘Missing Letter Puzzle’ as 8-bits/10.5-cc and 1046-bits/cover-character making them high-capacity steganography techniques with the opportunity for ‘Character Mapping’ and ‘Line Modification’ to be a variable amount based upon the chosen cover. In conclusion this source is useful in providing a great variety of text techniques in a digestible format, although lacking detail can be used as a reference or contents to research that is linked to ongoing practices that can be used to source information on techniques that can be explored for HTML practices.

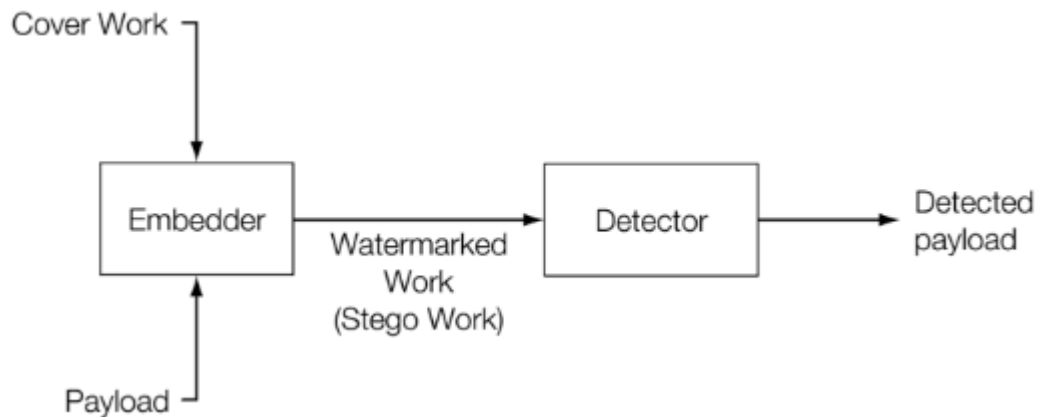
Digital Watermarking and Steganography by I.C, M.M, J.B, J.F, T.K

[11]

To better understand the function and purpose of steganography we can observe it in the context of watermarking which will provide further direction in terms of design. Reviewing the book ‘Digital Watermarking and Steganography’[11] by Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker gives good background, practical, and technical information that directly relates to this project’s potential applications so should be considered in high regard. The book primarily covers an independent point of view about specific methods of watermarking video and images with overlap into techniques which can be more widely applied to a variety of media

“This book is not intended as a comprehensive survey of the field of watermarking. Rather, it represents our own point of view on the subject. Although we analyse specific examples from the literature, we do so only to the extent that they highlight particular concepts being discussed.”[11]

First listing a variety of ‘19’ encoding methods for watermarking the book demonstrates examples in C++ within Appendix B. To later be referenced when referring to techniques, many of these techniques involve complex encoding to mask data within a video/image in a sophisticated manner which does not directly relate to text steganography but is supported from the first chapter onwards. A generic watermarking Figure typifies many methods and relates to all approaches used by this project’s algorithm. While also distinguishing between ‘covert’ and ‘overt’(C 1 – P5.) techniques indicating that steganography firmly relies upon the covert, while overt is left to encryption.



(Fig 1.4) Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Generic representation of a complete steganography system, 2008.*[11]

Security sporting its own definition in steganography as ‘*the ability to resist hostile attacks*’ in regards to watermarking, is split into three types: passive, active, and hostile(p55.).

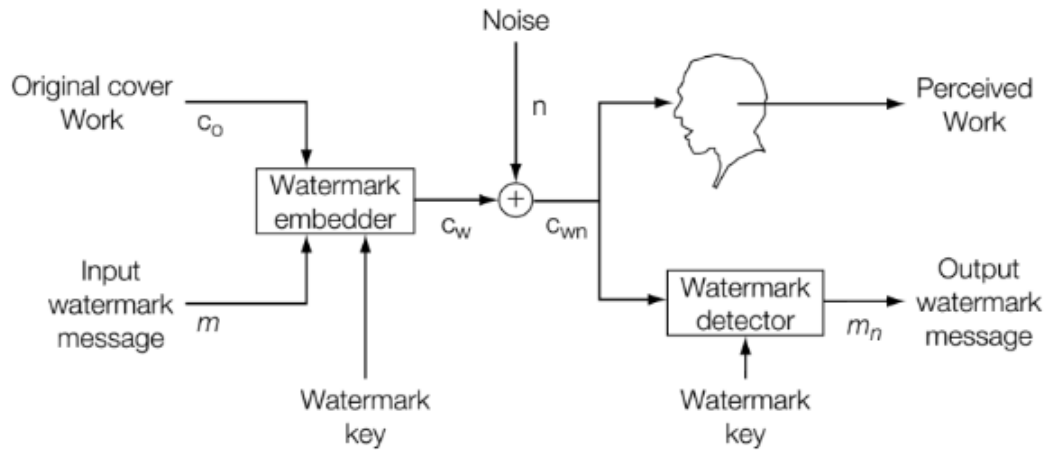
- Passive – Purely observational, the attacker lacks tools and can only observe the cover document to detect signs of steganography(most common)
- Active – The attacker will attempt to disfigure the steganography via tools or where they believe a message to lie using methods like: corrupting, deletion, and merging
- Malicious – The attacker is seeking to cause problems for either party and benefit themselves, this is most common in tactics that involve tampering, spoofing, and impersonation.

These warden concepts will be useful to evaluating any new steganography technique as a form of robustness and to bring a definitive terminology to the word ‘*security*’ when regarding steganography. On top of providing a security model for steganography this book also outlines the process of testing a scheme

“There are a number of blind steganalysis algorithms that can be used to test any steganographic method. To do so requires a large and diverse database of test Works that, ideally, would accurately model the conditions under which the steganographic algorithm is to be used. Such a database is divided into two sets, the training set and the test set.”[11]

These testing approaches will be used when testing this projects algorithm, each algorithm that is then comparable will be evaluated against these results. The essence of these practices concludes the first chapter and are listed on page 58.

When mentioning detection and embedding the book models cover text as ‘*High-dimensional media space*’ meaning that particular techniques and media used will often not be fully encoded but have ‘*regions of interest*’ which can better optimize approaches to detection and test against to create a robust algorithm.



(Fig 1.5) Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. Key based complex representation of a complete steganography system, 2008. [11]

This source is far more comprehensive but has a larger scope than other sources[10][9][8] it provides individual technical information that can be used to study and adapt existing approaches, many of which approaches lie within bit level[10] manipulation of images and transferable methods for both video and audio. This project will seek to build upon the text field by utilising the formal security and testing methodologies described within this source. These methodologies could also be applied by third party to other project algorithms, this would be a useful endeavour to catalogue the robustness of multiple researched approaches.

Four text based steganography research and methodologies[12][13][14][15]

Source A: Text Steganography in chat by M.H.S and M.S[12]

Source B: Text Steganography by Changing Words Spelling by M.S[13]

Source C: Experimenting with the Novel Approaches in Text Steganography by S.D, D.J and A.D[14]

Source D: A novel approach of secret message passing through text steganography by S.K and K.M[15]

The following pieces of research have been reviewed together as they all discuss a selection of text-based techniques. Each technique has unique aspects which have transfer to HTML steganography some of which have been mentioned within sources previously[9-10]

Text Steganography in chat by M. Hassan Shirali-Shahreza and Mohammad Shirali-Shahreza's[12] primary focus is using steganography in a text chat application setting and features several techniques which would apply to an SMS setting. Among more common techniques present in other sources Source A references rarer more unique techniques which arise from text communication. 'Spam texts' is mentioned as a potential vector for steganographic communication

“This method is based on the Internet. Based on the piece of information aimed for hiding, a number of spam emails are selected and some phrases appropriate to the concerned piece of information are chosen from among these letters, so that the concerned piece of information might be extracted from these phrases later.”[12]

Other examples given appear within R.K, P.T, and M. B’s ‘An overview of text steganography’[10] such as spacing techniques, syntactic methods, and feature encoding. On top of this the source refers to a more common less considered steganography of texting abbreviations, more of a widely understood form so less secretive.

Similarly spelling words wrong through abbreviation has similarities with other steganographic schemes as seen in *Text Steganography by Changing Words Spelling* also by Mohammad Shirali-Shahreza [13]. Separately from Source A, Source B proposes the use of misspelling to encode a message similar to how you may message in a rush or speak in short online using abbreviation and acronyms. The paper suggest that a bit(1) could be encoded by switching language styles between English(US) and English(UK), in example a ‘I’ could be encoded within a sentence if a word with alternative language spelling appears in English such as ‘center’ vs ‘centre’. This can be applied to many words and across a paragraph or longer document encode a full message.

“The extractor program will extract the data from the text. This program identifies the type of words in text by using the list of words having different UK and US spelling and saves the quantity of 0 or 1 in an arrangement according to the fact whether it is a US word or UK word.”[13]

This approach has limiting factors which are referred to in this paper, the largest limiting factor will be size. To encode a full message across a document using ‘1-bit’ only when an alternative word can logically occur means that this technique will not be very space efficient. On top of this this approach only has applications to documents which appear exclusively in English limiting the scope of potential communications that could be exchanged.

“On the other hand, in countries which their native language is not English and their English language knowledge is low, substituting US and UK spellings of words are not attract attention much, so this method is applicable in these countries.”[13]

American Spelling	British Spelling
Favorite	Favourite
Criticize	Criticise
Fulfill	Fulfil
Center	Centre
Dialog	Dialogue
Medieval	Mediaeval
Check	Cheque
Defense	Defence
Tire	Tyre

(Fig 1.6) Mohammad Shirali-Shahreza. Potential compatible words list,2008.[13]

A more sophisticated solution is offered within Source C *Experimenting with the Novel Approaches in Text Steganography* by Shraddha Dulera, Devesh Jinwala and Aroop Dasgupta[14] This paper focuses on letter grouping to improve space efficiency and encode multiple bit patterns. The process initially evolves from dividing ‘rounded’ characters from ‘straight’ characters, indicating if a bit is a ‘1’ or ‘0’ respectively each capital of a new sentence is then used with sentence distortion possibly being needed to fulfil a complete message.

“

“All birds can fly. Ostrich is a bird. Ostrich can also fly.”

Now, to hide the bit ‘1’, we can select the first letter of the first word i.e. ‘A’ which appropriately is the member of group B. However, we cannot hide the next bit which is also a ‘1’ using the first letter of the beginning of the next sentence – which is ‘O’ belonging to group A. Therefore, we modify the next sentence as follows:

“All birds can fly. This is a bird. Ostrich can also fly.”

”[14]

Source C further develops this idea by adding new letter groupings to further increase space efficiency; letters are now split into four categories:

Group ID	Group Name	Letters in group	Bits used
A	Curved Letters	C, D, G, O, Q, S, U	00
B	Letters with middle horizontal straight line	A, B, E, F, H, P, R	01
C	Letters with one vertical straight line	I, J, K, L, T, Y	10
D	Letters with diagonal line	M, N, V, W, X, Z	11

(Fig 1.7) S.D, D.J, A.D. Proposed letter groupings based on attributes2008.[14]

labelling this method ‘*Quadruple categorization*’, each of these groupings have a best case time complexity of $O(1)$ and a worse case of $O(N)$. Despite the progressive letter division, the evidence that steganography is being used is visually apparent, as malformed sentences to conform to this scheme will arise suspicion or at best seem less coherent.

Source D *A novel approach of secret message passing through text steganography* by Santanu Koley and Kunal Kumar Mandal [15] Improve upon schemes like Source C by firstly being compatible with multiple languages and having a greater complexity to stifle detection. Santanu Koley and Kunal Kumar Mandal propose a new steganography scheme in which characters are grouped by ASCII value (American Standard Code for Information Interchange) and are encoded within a document in a date format (DD/MM/YYYY).

The logic for translating a date into a value is:

Date = DD/MM/YYYY
Where M == Date.DD & N == Date.MM
$$[(M (M+1)/2) + N]$$

In example a value 'A,' within the ASCII table is 65(Decimal) to encode 65 we can use 6 as a day value and 11 as a month value (06/11/2020) Note that the year value does not weigh into this equation and is simply a cover. To decode the algorithm simply substitutes into the algorithm, the acceptable ranges of dates must stand and because of this there are limited combinations that will apply to this formular they are however varied enough to provide steganography encoding. The shortcomings of *Santanu Koley* and *Kunal Kumar Mandal* scheme stem from space efficiency and the frequency in which dates appear within a document, each date only encodes 1-bit so to fully send a message multiple dates will have to appear in the text. This is acceptable and accepted in a historical writing or event poster, this significantly limits application dependant on the level of secrecy required. The main motivation behind this scheme is to provide an esoteric and unique format

"This system is captivating the benefit of the subsistence of numerous unusual types of Steganographic approach in a variety of languages."[15] To this end the algorithm accomplished this, with concern shown towards the future of steganography being more closely tied to security; a common sentiment among researchers.

These four sources[12][13][14][15] all display an understanding of steganography in a broad scope and provide a unique scheme placed within the context of text steganography, many schemes which are frequently researched bear resemblance to techniques discussed within *Digital Watermarking and Steganography*'[11] by *Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker*. The four sources discussed although unique could be grouped into existing categories:

As described by: (Fig1.3) *R.K, P.T, and M. B's An overview of text steganography*[10]

Source A[12]: Synonym steganography

Source B[13]: Spelling of words

Source C[14]: Character and String mapping

Source D[15]: Character Marking

All of the above concepts relate to text steganography and as a result HTML steganography as HTML uses a text medium to produce code, many of these concepts reoccur and can be referenced to better understand mechanisms which rely off these basic concepts but add additional layers to better suit HTML or provide and aspect of security; an aspect which is also fed into by future work relating to encryption and other more secure forms of safety alongside secrecy.

Three HTML based steganography research and methodologies[16][17][18]

Source E: A Novel Text Steganography Technique Based on Html Documents by M.G/[16]

Source F: Novel Steganography over HTML Code by A.O, K.E, M.F and E.A/[17]

Source G: A New Method for Steganography in HTML Files by M.S.S/[18]

The following sources **E-G** are grouped together as they discuss the field directly related to this project HTML steganography, by reviewing each source in this manner we can build a research base(see 1.4) that will further support personal research and comparison. These four sources were originally referenced within the **Initial Document** of this project and have been reading and research closely associated with the finished algorithm.

Source E:

It is not uncommon for aspects of HTML itself to be used to encode, the difficulty in these approaches is to maintain the integrity and readability of the code. Changes that are low impact to the result of execution are not always low impact visually and vice-a-versa, Mohit Garg's paper 'A Novel Text Steganography Technique Based on Html Documents'[16] clarifies an attributes-based technique and takes this a step further by introducing a key generation scheme to introduce an aspect of reversibility to the encoding. The paper is solely focused on this unique method, and it fully documents the details with example of the encoding and decoding procedure and offers some background information on the framework of the scheme with relevant reference to the field of text steganography. The approach used is closely related to the project as it encodes into html and fulfils the objective attributes which are required for a good steganography algorithm, because of this the source was used as a point of inspiration and a waypoint that covers HTML steganography.

The middle and late portions of the source fully describe this attribute and key based scheme M.Garg has produced, in brief the scheme involves ordering attributes according to a key which is generated to determine which pairs of attributes produce a '1' or a '0'

"The proposed technique uses the html tags and their attributes to hide the secret message. It is based on the fact that the ordering of the attributes in the html tags has no impact on the appearance of the document. This ordering can be used to hide the secret messages efficiently."[16] By using key generation the program can determine the primary and secondary attribute meaning compatibility when reversed assuming the key has been written, an example of a valid pair looks like:

[16]

```
1<html>
2<head>
3    <title>Canary Birds</title>
4    <meta name="author" content="Peter Miller">
5    <style>
6        .bigText{ font-size:14px; font-weight:bold; }
7    </style>
8</head>
9<body text="#000000" bgcolor="#FFFFFF" link="#FF0000">
```

Where attributes are marked in ‘*red*’ and all attributes used as a primary or secondary key appear ‘bold and underlined,’ note that lines containing single attributes are skipped as they cannot produce a key. *M.Garg* describes the decoding process

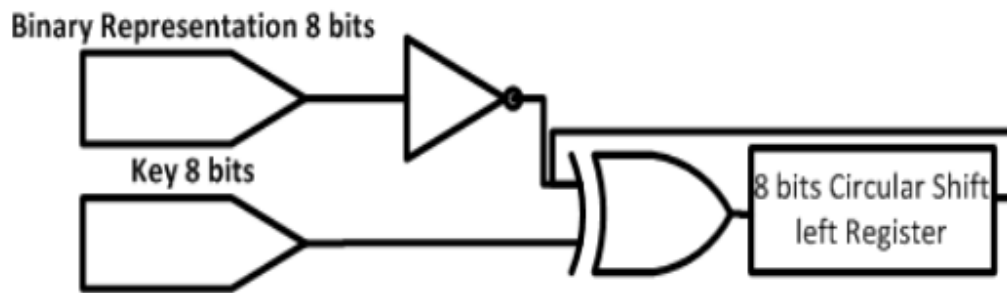
“The extraction component is quite simple. Firstly, it scans the document to find the attribute pairs that hides a bit, using a similar procedure as used for hiding. Once it finds the attribute pair, it compares the positions of the attributes according to the key file.”[16] Once the attributes are compared to the key file it is a simple process of collecting bits together and converting from binary to spell a message.

This technique has great capacity to hide messages as many HTML documents contain a great number of tags and attributes allowing larger numbers of bits to be encoded, one area that could be problematic is best fitting the document to the encoded text. Some programming standards demand that attributes be in order of priority or sensibility such as the idiomatic-HTML[33] style guide, conforming to this style and encoding could cause complications when using this scheme. Despite this it is an effective scheme which fulfils the requirements of the projects objective so was closely read and researched as an example of what would ideally be the outcome of the project.

Source F:

‘Novel Steganography over HTML Code[17]’ by *Ammar Odeh, Khaled Elleithy, Miad Faezipour, and Eman Abdelfattah* seeks to design an algorithm that incorporates and element of security while also maintain elements of secrecy, this was direct inspiration alongside a recurring theme within the conclusions of multiple research pieces to incorporate a security factor within the personal project. By adding additional security features these applications are better suited to supplement encryption and provide ease of mind when used in a serious/critical setting. Aside from a concise introduction and background this paper focuses solely on the designed algorithm and some of the nuances in dealing with HTML type documents. The paper discusses a unique algorithm *‘In this paper, we employ cryptography and steganography techniques to pass secure information.’*[17] comprised of both types the method of steganography will be covered first. The algorithm uses letter frequency on the plaintext to determine the most occurring letter, this letter is then assigned a ‘1’ and the next most common is assigned a ‘0’ by doing this the method is able to comprise a list of characters that do occur and don’t occur and group characters into binary saving on space for characters which are not as common. This technique is not dissimilar from this personal project which uses *‘Letter frequency analysis’* to reduce space.

A simple encryption process is then followed once the plaintext has been passed through the decided bit filter, this process consists of applying a key against the binary data through XOR gate with an LSL(Logical shift left) and repeated.



(Fig 1.8) Novel Steganography over HTML Code. Simple encryption scheme, 2015. [17]

The data is then converted to ASCII and ready to be inserted, the paper describes insertion into a comment on the codes page this is where the encoding will be read from, standard HTML comments do not appear when a page is viewed only when the source is inspected. For this reason I believe that the intent of this steganography scheme is to only embed a message on a website online and not distribute the code itself, as if it were inspected it would be immediately clear that the comment '`<!--UIFNWHTK-->`' is attempting to hide something, this goes against the aim of one of my objectives as the algorithm should maintain secrecy even when inspected or decompiled. The decoding process is the same as the encoding process with the decryption being run with the key to reveal the initial assigned bits.

In conclusion the paper summarizes the results and what is achieved by the algorithm

- 'Language independency' can be applied regardless of language
- 'Algorithm transparency' level of hiding applied to the data
- 'Hidden ratio capacity' number of bits that can be encoded per page
- 'Algorithm reusability' can be applied across a variety of files
- 'Algorithm robustness' restricts change to cover text

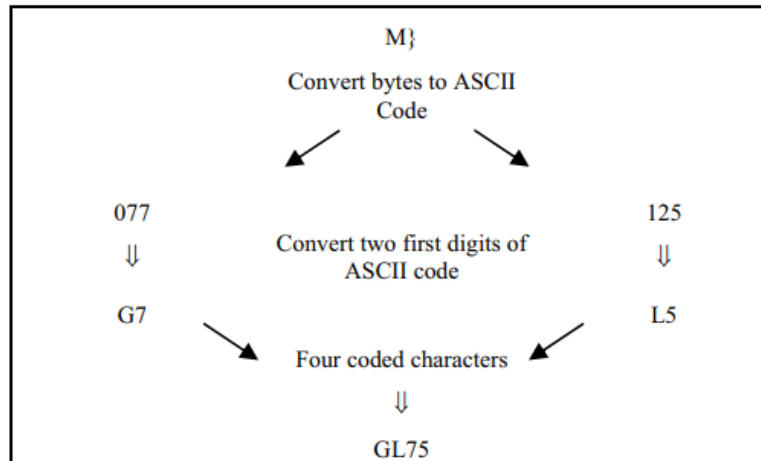
These evaluation metrics will be useful to show completeness of an algorithm and are closely tied/described by this projects objectives.

Source G:

A solution that better masks the technique when viewed as source is documented in *A New Method for Steganography in HTML Files* [18] by Mohammad Shirali Shahreza, this paper covers M.S.S's unique steganography technique involving the use of ID attributes with reference to other existing HTML approaches much like other papers.

Many ID attributes exist within HTML documents, similar to class files they are used as reference when styling with a key difference being ID attributes must be unique. They must possess a unique name which is non-standardised, many documents simply use ascending numbers or a series of characters that remind the style guide the purpose of the object. This makes ID attributes a good target vector for steganography as they exist frequently, can be a combination of many different types of character, and are inconsequential to the programs operation

(assuming they are referenced appropriately). Or as *M.S.S* describes “One of the attributes that exists in almost all tags on HTML pages is the ID attribute. Based on this attribute, each tag is given a unique ID. Our purpose in this algorithm is to hide information in this tag attribute.”[18] However the characters cannot simply be inserted into the ID as this would prevent source text secrecy, *M.S.S* describes a technique in which each bit of the plaintext ID is split into separate ASCII table values[34], once these values are obtained the leading two numbers are used to convert back into ASCII text and are concatenated into a finished ID. This process is described by diagram:



(Fig 1.9) A New Method for Steganography in HTML Files.ID conversion process,2005.[18]

A similar process can then be followed to decode each ID once scanned, introducing a non-apparent terminator ID can also help when implementing this scheme. This scheme when applied to **Source F**'s success metrics[17] shows superior performance, the algorithm can be used regardless of language since ID is an arbitrary value, ‘Algorithm transparency’ as the source can be observed without suspicion(dependant on Hidden ration capacity). The ‘Hidden ratio capacity’ will vary depending on the comfortable level of suspicion as longer ID values will become increasingly easy to detect, this algorithm may struggle with ‘Algorithm reusability’ as the number of ID’s can vary greatly, larger professional pages are likely a good candidate for this algorithm as they have larger style sheets. Lastly algorithm robustness is high but still not as high as it could be ID is still a changed attribute within the document, to maintain full robustness in this manner there would have to be no visual change to the document. This factor was a key motivation behind this personal project, as having a high level of robustness and secrecy would allow all normal operation of the file to be observed without suspicion.

Four Closely Related Research Papers[19][20][21][22]

Source H: *Enhanced text steganography in SMS* by K.F.R/[19]

Source I: *Digital Steganography: Hiding Data within Data* by D.A/[20]

Source J: *Information Hiding: A New Approach in Text Steganography* by L.Y.P and B.D/[21]

Source K: *A Review on Text Steganography Techniques* by M.A.M, R.S, Z.S, and M.K.H/[22]

These four sources **H-K** are closely related to the method of encryption used by this personal project and have served as background to the chosen technique. Each of them has commonalities and specifics regarding blank space steganography, this personal project fits into the following sub-divided areas: Text Steganography, HTML Steganography, Character Marking, Blank space Steganography. These sources accurately describe concepts within these categories and are used to better define and contribute towards the explanation and planning of this projects chosen technique.

Despite being titled as '*Enhanced text steganography in SMS*' Source H contains information relating to techniques involved in communication in multiple formats, this comes to include HTML as the format is most used to display online web pages. HTML is first mentioned in *Sect 2.4.[19]* where tagging techniques are listed alongside white space techniques. The whitespace technique shown does not conform to the objectives and in a HTML, context would be transformative to the document to a noticeable degree due to spacing between syntax, this personal project sought to build upon these basic whitespace concept to provide a scheme which is in essence '*invisible*' inside and outside of the source editor. Other techniques such as line spacing and tabulation are mentioned but not in details directly related to the personal project. *Digital Steganography: Hiding Data within Data[20]* by Donovan Artz Goes further in depth on '*Data ordering*' in which the organisation of data can lead to the encoding of information

"*Each permutation of a set of objects can be mapped to a positive integer. This mapping can then be used to encode hidden data by altering the order of objects that are not considered ordered by the carrier medium.*"[20] the paper mentions re-encoding while using data ordering can ruin encoding however this will only apply to documents that are not recognised as encoded and thus sanitised before, the personal project has accounted for this and will always work off the base cover text fed into the application, on top of this each new run of the algorithm nullifies possible encoded areas to reproduce a '*clean document*'. The whitespace component is more comparable to '*Information Hiding: A New Approach in Text Steganography*'[21] by Yee Lip Por and Delina Beh Mei Yin by using whitespace approaches we are able to make effective steganography inside and outside of documents, assuming spacing and tabulation is correct we can also maintain proper functionality of the document keeping in line with requirements **Source J** describes the evolution of whitespace techniques and creates a unique scheme by using a combination of spacing, tabulation and whitespace. This technique alone does not provide an aspect of

security and if mapped 1:1 to ASCII will be more trivial to decode.

“Currently, manipulation of whitespaces seems beneficial and has its potential in information hiding because whitespaces appear in a text documents more than the appearance of words. It is even an advantage when no one will know that a blank piece of document is actually vital secret information.”[21]

This personal project has taken inspiration from this papers approach to whitespace encoding but additionally seeks to add an aspect of Security/Obscurity to prevent a 1:1 relation, most commonly ASCII equivalent encoding will produce a common bit pattern which can be used to detect and decode white space steganography techniques. In future work this paper would improve robustness of this algorithm by having more permanent encoding or better detection of encoded documents.

“The future work should be focused towards optimizing the robustness of the decoding algorithm.”[21] This was a key motivation to implement greater robustness into the algorithm and has allowed review of other potential robust whitespace methods. *A Review on Text Steganography Techniques* by M.A.M, R.S, Z.S, and M.K.H[22] goes further to describe desirable attributes of steganography as a triple: Robustness, Capacity, and Security. Each playing an influential role in judging the effectiveness of a steganographic algorithm. Because of the nature of secrecy, it is difficult to produce an algorithm which exhibits all the attributes to their fullest or as the paper describes

“These are the most influential factors that determine the efficacy of a steganography setup. According to its use, there are certain specific requirements for managing a number of steganographic designs. Steganography and watermarking have these attributes for data embedding. However, a common trade-off is between the size of secret data and the quality of the stego files.”[22] Robustness also contributes to the protection against attack as many techniques may aim to disrupt secret communication, with great robustness it is harder to stop communication.

Four Sources Describing Use Cases[23][24][25][26]

Source L: Steganography and Its Applications in Security by R.D, P.J and L.G[23]

Source M: Digital Watermarking and Steganography: Fundamentals and Techniques by F.Y.S[24]

Source N: Information Hiding steganography and Watermarking Attacks and Countermeasures by N.F.J, S.J and Z.D[25]

Source O: A compression-based text steganography method by E.S and H.I[26]

It is worth considering the potential use cases of various steganography techniques as much of the field becomes saturated with encryption techniques as a primarily secure method of communication **Sources L-O** describe, research, and evaluate use cases of steganography, with the most common ‘*Watermarking*’ being a recurring theme throughout all four sources.

‘*Steganography and Its Application in Security*’[23] by Ronak Doshi, Pratik Jain, and Lalit Gupta provides a varied in-the-field view of steganography techniques and uses specifically within a corporate and domestic setting as well as potential malicious use

when secrecy is applied to nefarious activities.

“Steganography can also be used for corporate espionage by sending out trade secrets without anyone at the company being any the wiser. [7] Terrorists can also use steganography to keep their communications secret and to coordinate attacks. All of this sounds fairly nefarious, and in fact the obvious uses of steganography are for things like espionage. But there are a number of peaceful applications.”[23]

The source continues to explain that features such as watermarking have usefulness in software, for example CD's, Videos, and Games (as a means of ownership detection) and Documents, Code (To prove the Author or ownership). When applied to the personal project Code Ownership and Authorship is best suited the algorithm; already using HTML code as cover text this will allow programmers to secretly mark their code. As is trending in the field the source concludes with mention of using steganography combined with cryptography, using inference it is assumed that this will make a 'hackers' job harder as the vector for hacking will be somewhat masked and secure.

‘Steganalysis is a process in which a steganalyzer cracks the cover object to get the hidden data. So, whatever be the technique will be developed in future, degree of security related with that has to be kept in mind. It is hoped that Dual

Steganography, Steganography along with Cryptography’[23]

‘Digital Watermarking and Steganography: Fundamentals and Techniques’[24] by Frank Y. Shih further elaborates on digital watermarking making a clear distinction between steganography as a concept (in a vacuum) and digital watermarking noting multiple differences which cannot directly be applied between the two, for example it may be desirable to have an obvious watermark in which case steganography is not needed.

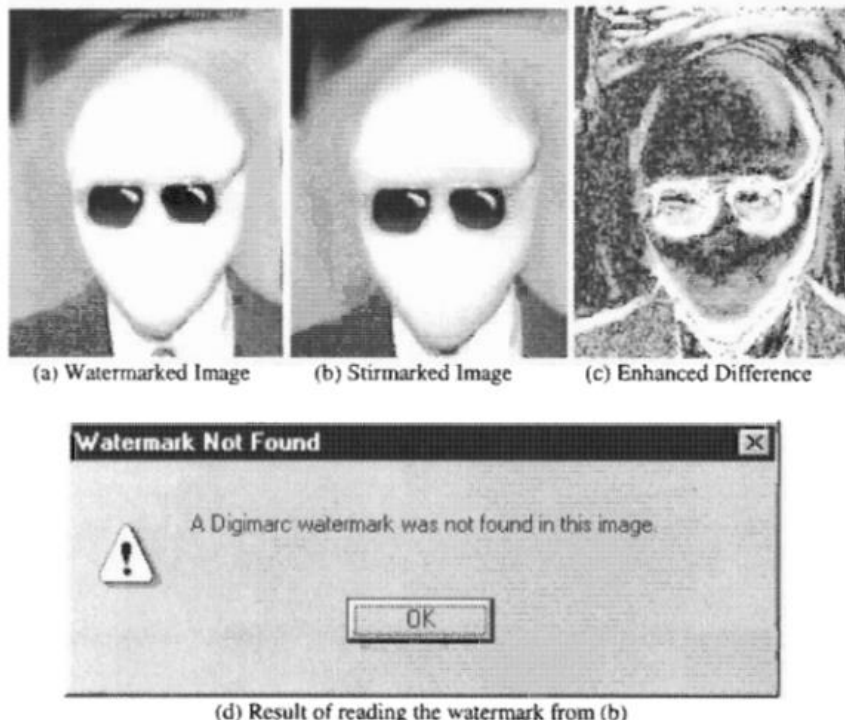
“Watermarking is often used whenever the cover image is available to users who are aware of the existence of the hidden information and may intend to remove it.”[24]

The idea that watermarking and steganography are separate isn't a new idea. Images, Videos and Documents have been watermarked since the advent of the formats. This document for instance will have a form of watermark such as a name and institution and will not be hidden or writable when in PDF format. As most watermarks seek to protect the works of an author it is no surprise that this has leant itself to corporate and other monetary incentives, however this is not the sole purpose of a watermark as an author may simply want recognition to be able to stay informed on how others have further developed their work. This is quite common with repository and version tracking suites such as 'Github'[35] allowing changes to be reviewed and progress/forks to be reported to the original owner. *‘Information Hiding steganography and Watermarking Attacks and Countermeasures’[25] by Neil F. Johnson, Zoran Duric, and Sushil Jajodia discusses the details of watermarking but also attacks alongside countermeasures, by understanding these components creating a steganography algorithm around ‘robustness’ is more achievable. One simple method of attack is ‘Distortion’ where data is simply overwritten in attempt to remove a watermark, this may make the cover content un-readable but methods involving distortion can be tweaked for more favourable results depending on the*

media **Source N** describes distortion in (sect 3.2)

“The methods devised for disabling embedded data are not intended to advocate the removal or disabling of valid copyright information from watermarked images, as an illicit behavior, but to evaluate the claims of watermarks and study the robustness of current watermarking techniques. [28] Some methods of disabling hidden messages may require considerable alterations to the stego-image.”[25] The source goes on to prove this quote by testing various distortion techniques using a consistent testing methodology.

1. Use existing images and create images for testing.
2. Embed messages and watermarks in the images.
3. Verify the resulting stego-images contain the embedded messages.
4. Compare the resulting stego-images with the original images.
5. Look for patterns when using different messages and/or different images (*chosen message attack*).
6. Manipulate the images with simple image processing techniques and check the embedded message.



(d) Result of reading the watermark from (b)

(Fig 1.10//1.11) Testing methodology and distortions from using ‘StirMark’ distortion technique, 2016. [25]

As can be observed from Figure. 1.10 and 1.11 it is possible to remove a digital watermark without significant impact on the resulting media. Displaying the importance of secret watermarks but also the limitations of online verification as the result to a human would be easily recognisable as the same work just not verifiable. Aside from steganography there exists more robust approaches to adding watermarks which are visible: gradual watermarks (watermarking in areas of low contrast to draw less attention), fingerprinting (using salient feature points such as edges or using Normalized Cross-correlation),

Affine Transformations(masking a watermark by using values to stretch scale and position a watermark) described in detail p79-91[25]. These methods rely of transformative measures to disguise a watermark, the goal of some transformative measures is to have a third party unaware that a watermark exists in the first place very similar to steganography, this may be relevant in future work as it is sometimes evident based on the source that a watermark is present making the secrecy, robustness a greater need. Similarly, '*A compression-based text steganography method*' by *Esra Satir* and *Hakan Isik*[26] describes a unique method of watermarking using compression a technique which has many parallels to distortion. This source focuses on the motivation, related work, and methodology behind the algorithm, uniquely using LZW(Lempel–Ziv–Welch) a compression method published in 1984 and predominantly used within Unix[36]. The operation of embedding closely follows this compression scheme, the general operation of which is when large and frequently occurring repeated data is progressively replaced with an indicating code and stored in a dictionary this makes the compression '*lossless*' and can be easily decompressed to return the initial input. *Ersa Satir* and *Hakan Isik* utilize this compression and manipulate the codes which are stored within the dictionary to encode a specific character, in example this could be done by appending a certain number of '@' symbols in a compressed email address. The usage of compression is relevant to the use cases of steganography as they can both be applied in a useful manner, compression is a highly demanded and ubiquitous technique, with methods such as described in **Source O** we can introduce a watermark every time we compress a file.

Concluding information and Supporting Sources(additional reading)[27][28][29][30][31]

All of the listed sources provide additional reading aside from sources [8]-[26] and information that is relevant but not immediately unique to what has been researched already. These reviews will provide some concluding information and useful background into other routes research could be carried for alternative topics. '*A Comparative Analysis of Arabic Text Steganography*'[27] compares numerous Arabic text steganography approaches some already researched and referenced[10] '*Dot method*', '*Diacritic Method*', and '*Sharp Edges Approach*'[27] each of which are specific to Arabic text as it displays unique characteristics. This source is useful to evaluation and comparison of steganography techniques working within the same specific field. Evaluation is carried out by a capacity metric equation three tenants of a successful algorithm robustness, security, capacity. The paper found that overall, every algorithm suffered from a low capacity, especially becoming apparent when specific rare features of text are chosen as there is no guarantee those attributes will be frequent enough to encode with. '*Steganography in Digital Media: Principles, Algorithms, and Applications*'[28] is a course text which covers a broad range of steganographic concepts, from the very origin and history of the word to the comparison between the popularity of each vector for steganography. While also

covering topics not directly related but partially relevant because of the technology or technique used. In example CCD and CMOS sensors are used to describe colour interpolation

“To form a complete digital image, the other two missing colors at each pixel must be obtained by interpolation (also called demosaicking) from neighboring pixels. A very simple (but not particularly good) color-interpolation algorithm computes the missing colors as in Table 3.1. There exist numerous very sophisticated content-adaptive color-interpolation algorithms that perform much better than this simple algorithm.”[28] This source serves as additional reading to connected fields that interface with steganography and is a useful source for related research. *‘Trends in steganography’*[29] Is a review article covering culture trends in steganography from becoming a malicious tool for malware to its uses in networking. This source also builds upon the concept of steganography as separate from watermarking by breaking down individual factors which contribute to identifying each.

Table 2. Comparison of characteristics of steganography and watermarking.

	Watermarking	Steganography
Goal	Protect the carrier	Protect secret information from disclosure
Secrecy	Invisibility or perceptual visibility depending on the requirements	Embedded information is “invisible” to an unaware onlooker
Type of robustness	Robustness against tampering or removal	Robustness against detection
Characteristics		
Effect of signal processing/ random errors/ compression	Must not lead to the loss of the watermark	May lead to the loss of hidden data
Type of carrier	Digital files—audio, video, text, or images	Any service, protocol, file, environment employing digital representation of data

(Fig 1.12) Comparison between Watermarking and Steganography characteristics, 2014.[29]

As described by the above (Fig 1.12) Steganography focuses on secrecy, robustness and disclosure of information and can be more widely applied to many media formats. This information corroborates what is described by *Frank Y. Shih* *‘Watermarking is often used whenever the cover image is available to users who are aware of the existence’*[24] *‘Survey Paper on Steganography’*[30] and *‘Computer based steganography: How it works and why therefore any restrictions on cryptography are nonsense, at best’*[31] by *Namrata Singh and Elke Franz, Anja Jerichow, Steffen Möller, Andreas Pfitzmann, Ingo Stierand* respectively both attempt to review steganography as a

field. In the second source steganography is more closely reviewed alongside cryptography as the main competition in security. The paper argues that the variety of digital formats have given way to new and unique steganography methods and perhaps hiding within unknown formats can also be effective.

“In the future, messages, e.g. speech, text or pictures, will be transmitted digitally since this is cheaper, more perfect and more flexible. It is possible to hide messages, which are of necessity much shorter, nearly unrecognizable for outsiders in such digitized messages.”[31] This paper specifically vouching for the effectiveness of the combination of techniques.

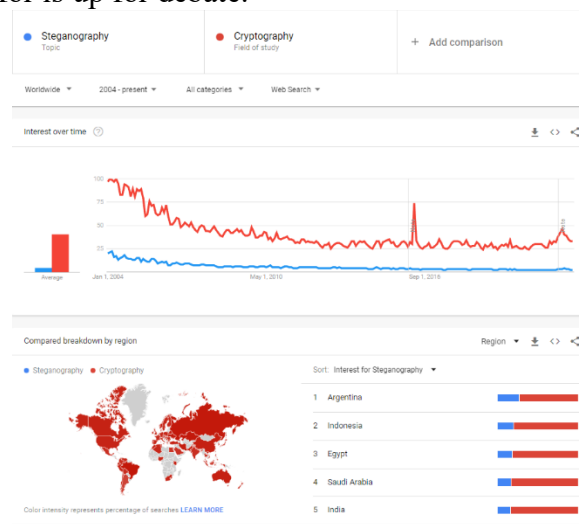
(See Appendix A.)

1.5 Research Base

After careful review of a selection of relevant research, development, and review papers relating to the field of steganography, this literature review has come to multiple conclusions and considerations that are currently documented in the field that can be applied to this project and serve to better guide and prove the validity of discussion and future work relating to steganography.

Firstly the exact definition of steganography has been better clarified through a multitude of sources, as described by *Namrata Singh*

“Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The word steganography combines the Greek words steganos, meaning "covered, concealed, or protected", and graphein meaning "writing".”[30] a definition which is mostly agreed among other referenced sources which provide background or history. Steganography although understood when explained is a term not frequently used, whether this is due to the greater need for security through cryptography or the intentional lack of discussion to preserve techniques to look for is up for debate.



(Fig. 1.13)Google trends data 2004-2022 for steganography vs. cryptography[37]

Secondly the distinction of steganography techniques and the usefulness of a combined technique steganography in synergy with cryptography has proven a consistent theme throughout research. Most notably in steganalysis papers the weakness of watermarking becomes apparent when using distortion techniques, instead recommending the use of cryptography and steganography to increase this security factor[23]. By employing cryptography we can also ensure that if an embedded message is found it will not be immediately readable, this will however only work in cases where communication has been previously established with secure key exchange.

Additionally the success metrics(robustness, security, capacity, concealment) used to evaluate steganography techniques will prove useful in comparing and contrasting different techniques as they represent important factors in each unique scheme.

Robustness will account for the permanence and resilience to attack, in example if a distortion technique is applied what data is changed vs unchanged and is this data valuable, or what happens if the data is re-embedded?

Security is similar but relates to the purpose of the malicious party, would a bad actor be able to read the embedded data? How difficult is it to automatically detect the embedded information.

Capacity the amount of bits that can be embedded per bits/per word/per line, many techniques struggle with this attribute as unique characters or elements are relied upon to indicate embedding.

Lastly concealment the detectability of the embedded data, visually or digitally it may be possible to detect a suspicious file from appearance or measured noise in comparison to the cover data described in detail as steganalysis in *Information Hiding steganography and Watermarking Attacks and Countermeasures* by N.F.J, S.J and Z.D[25] as many techniques do not incorporate security, any embedded data will be trivial to decode.

Each of these factors work in balance with each other in most normal cases by increasing capacity the technique has a larger embedded region making detection more likely thus limiting concealment, similarly robustness is affected by capacity as distortion techniques will be more likely to replace embedded regions.

The broad range of research techniques documented and those techniques similar to this projects approach has helped influence the direction of this project while giving background and benefits/drawbacks of approaches similar to those involved in this project. The sources closest to techniques used within this project are under the literature review heading Four Closely Related Research Papers[19][20][21][22] and focus on replacement, whitespace and shifting of text or HTML as these are the techniques chosen for this project.

The primary influence for this project are sources that were reviewed early in development of the project, these provided great inspiration and allowed an

exploration of complexity in a very varied field. These sources are referenced as and are also present in **Initial Report**:

- Garg, M. G. (2011). *A Novel Text Steganography Technique Based on Html Documents*. [16]

- Odeh A., Elleithy K., Faezipour M., Abdelfattah E. (2015) *Novel Steganography over HTML Code*. [17]

- Shahreza M.S. (2007) *A New Method for Steganography in HTML Files*. [18]

In addition to my initial research additional sources should be credited as primary influence following this current literature review:

- *Digital Watermarking and Steganography* by I.C, M.M, J.B, J.F, T.K [11]

A comprehensive background and great general coverage of the topic area, easy to reference and includes most details.

- *Information Hiding steganography and Watermarking Attacks and Countermeasures* by N.F.J, S.J and Z.D [25]

Information on specific hardware techniques and great level of comparative details, also provides information from the adversarial approach and steganalysis.

- *An overview of text steganography* R.K, P.T, M.B [10]

Broad selection of relevant and explained techniques with framework for evaluation used to grade and categorize each in terms of efficiency, capacity, and security.

1.6 Tools, Environment, and Resources

This section is dedicated to the setup used to develop the project to ensure reproducibility, this section also includes libraries used along with versions as the UI library and directory layout is heavily reliant on this. Resources relating to using various software and other documents that have informed development are also included.

•Python 3.8 [38]

The chosen language for this project, chosen because of easily understandable syntax and script like applications as this project would require heavy automation. Version 3.8 was specifically selected as it proved compatibility with many builds of PyQt5 which would be a required component for UI. A plentiful amount of documentation and plugins also have compatibility with this version, this helped in planning

development as certain components could be substituted for alternative methods. This version would also come with libraries that would prove useful themselves, the in-built Python 3.8 libraries used are as follows:

- ‘*itertools*’

Itertools provides advanced iteration methods which can be used to better filter and specify iteration conditions with no further code such as: `itertools.zip_longest` `.zip_shortest` this iterates to the longest/shortest value and will only exit when this value meets zero. `itertools.cycle` Saves a copy and repeats through iterating elements indefinitely. `itertools.compress` Iterates through elements filtering duplicates that evaluate to a true value Among other useful methods which were applicable to the development of this project.

-‘*string*’

For all variations of strings both input and output in the algorithm, this library became essential in compatibility of a variety of user input and subsequent string processing and passing.

-‘*time*’

Minor user experience and quality of life improvements to the operation of the user interface, mainly used for the included ‘`time.sleep()`’ functionality to improve readability and usability.

Resources online were also referred to in order to better meet time constraints and follow appropriate coding conventions:

Python 3.X docs: (<https://docs.python.org/3/library/index.html>)

Consulted to explore functionality of libraries and keywords used, to gain a deeper understanding of code used.

PEP8 style guide: (<https://peps.python.org/pep-0008/>)

Followed style guide for the project, keeps code standards consistent and enforced by default within PyCharm[39]

StackOverflow: (<https://stackoverflow.com/questions/tagged/python>)

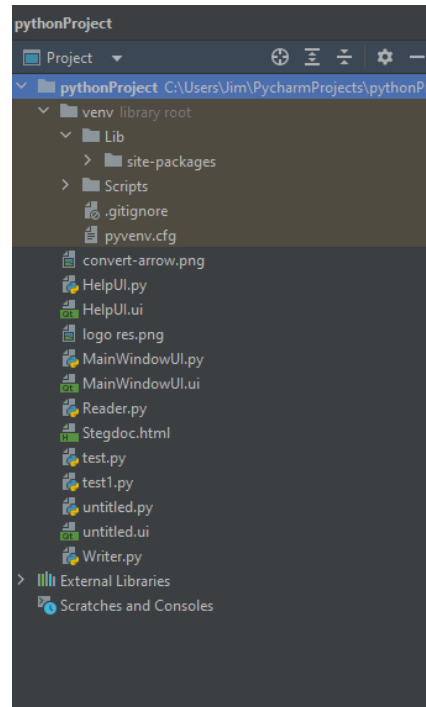
Used to investigate and debug problems within code and explore alternative approaches

(None of this project’s specific issues were fielded through StackOverflow)

•**PyCharm**[39]

Multi-functional IDE providing spell checking, version tracking, debugging, code suggestions, error and warning handling, file management and refactoring. All of which were applied to development of this project. This tool also helped in

preserving file versions and maintain a productive environment, after completing compilation each version was exported as their own `.py` and uploaded to the project GitHub[41] ensuring the changes could be properly tracked as the project evolves. PyCharm also benefitted the project in ordering the file system, both debug versions, compiled versions and other miscellaneous files such as the HTML input were kept in an appropriate structure allowing simultaneous changes to each.



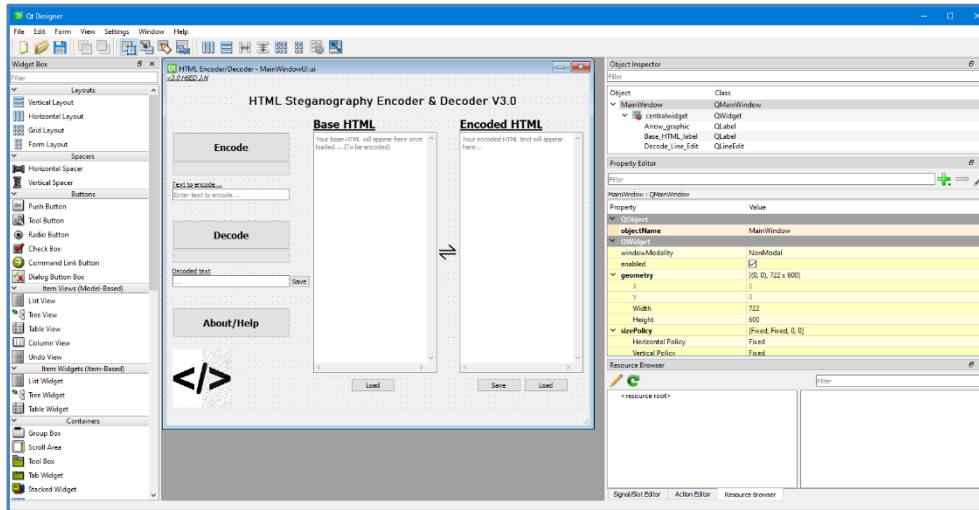
(Fig. 1.14)Development branch using PyCharm's file system[39][41]

PyCharm also showed high compatibility with PyQt5, allowing tools of the package to be used and saved within the environment while making changes to code. This allowed for progressive development and live debugging.

•PyQt5[40]

This library is primarily responsible for the projects UI, other options such as `'TKinter'`, `'PySimpleGUI'` and `'wxPython'` were a consideration but did not all offer the tools and variety of elements present in PyQt. Importantly it was important to experiment with UI and create layouts which would augment the algorithm to be more user friendly, PyQt was far more suited to this as it featured two notable tools `'QTDesigner'` and `'pyUIC'`.

Both of these tools provided a graphical user interface to create and interface which would then be translated to code to be further tweaked



(Fig. 1.15) Main workspace of QTdesigner editing the main window[40]

After saving using QTdesigner the file is converted from ‘.ui’ to ‘.py’ to be readable by the project using pyUIC a quick translation script. Once this file has been converted the ‘.py’ file can be linked to functionality such as buttons, text input and output; these of course can also be changed using regular python functions too.

As a primary resource this project referred to the tutorials point guide for concise explanation of various tools and functions available in PyQt5.

(<https://www.tutorialspoint.com/pyqt5/index.htm>)

Video resources were also used to connect functionality of the algorithm code to the UI so that it performed in a live and interactable manner.

Connecting user input: (<https://youtu.be/REQsOjJBwbQ>)

•GitHub[41]

Version control and tracking was especially important in documenting the development of this project as the algorithm evolved over time displaying new functionality and methods in each version. GitHub was chosen as it is among the most widely supported and documented repository website with CLI and advanced file handling as well as the ability to assign rights to the repository to allow collaboration and sharing. Also acting as a general project repository for documents and files ordered in a sensible manner to ensure all contents were available and accessible to be written about, written to, or linked.

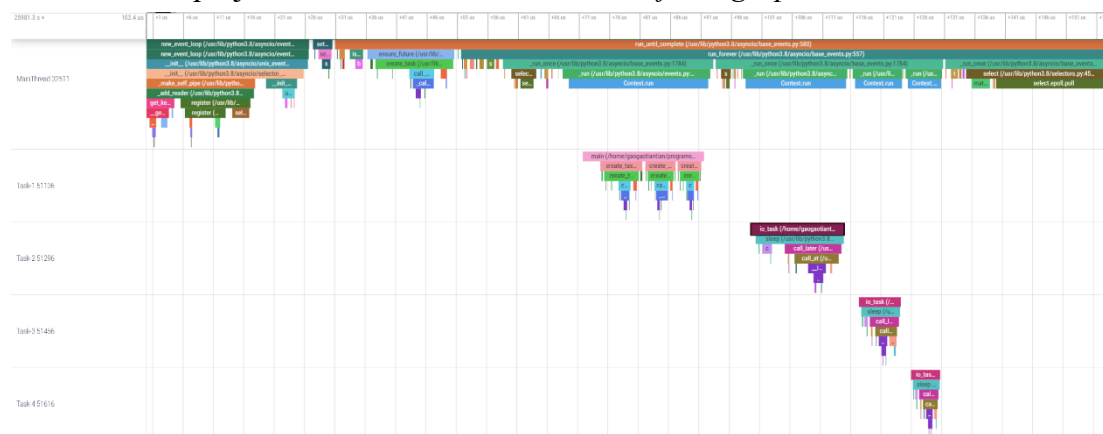
(This projects repository link is available at: <https://github.com/Ju5t-j1m/Y4project>)

•HTML viewer online[42]

The resulting HTML text when displayed as a page or plaintext is important to the operation of the algorithm and can be used as an evaluation condition, for this reason it was essential that the project have a reliable way to view and edit HTML in real time. Opting to use an online website with an in-built editor and viewer has allowed this project to tweak aspects of the embedding to see if it provided any noticeable differences from the cover text and verify that operation of the tags themselves remains consistent. Resulting embedded text can simply be copied from the resulting window and displayed when ran through the websites processor.

•Viztracer[46]

Viztracer is a visual code profiling tool or described by the author “*VizTracer is a low-overhead logging/debugging/profiling tool that can trace and visualize your python code execution.*”[46] This project has used this tool to profile and debug all functions from the user interface, a feature not fully supported by PyCharm as PyQt5 is an external library. Many views can be used to display code statistics; this project made use of the default and ‘*flamegraph*’ views.



(Fig. 1.16)Main view of Viztracer/Vizviewer used to profile a program[46]

Example encoding decoding test websites:

https://web.ics.purdue.edu/~gchopra/class/public/pages/webdesign/05_simple.html(simple)

<https://www.wikihow.com/Create-a-Simple-Web-Page-with-HTML>(complex)

<https://www.york.ac.uk/teaching/cws/wws/webpage1.html>(simple)

<https://learn.microsoft.com/en-us/training/modules/build-simple-website.html>(complex)

End of Chapter 1

Chapter 2

Algorithm and System development

Following on from the tools, resources and, environment used to develop this project's algorithm was constantly developing spanning three major versions[41] each with unique differences/features. This chapter will document the methods used, data collection, processing, UI/presentation, and output resulting in the final algorithm's development, documentation, and evaluation.

Time and Cost(See. Appendix A: Time Cost Gantt Chart)

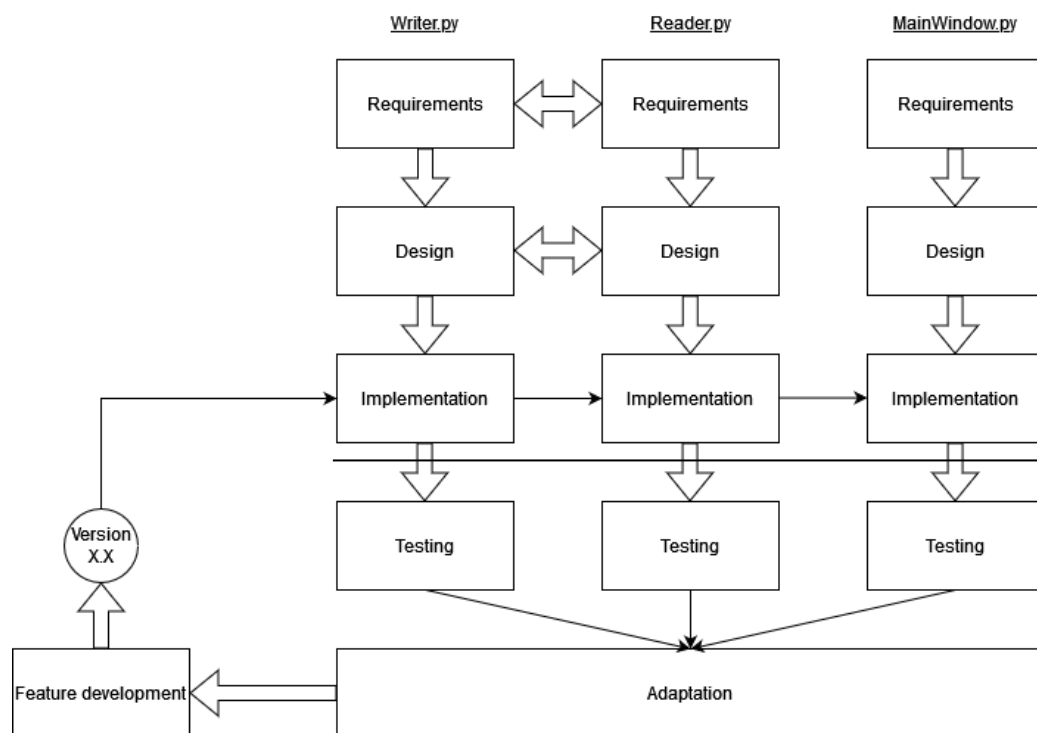
Table of Relevant terms

Term	Synonyms	Definition
Cover text	Cover doc/plain HTML	Text to hide data within
Stego text	Embedded doc/Encoded doc/stego file	Document with a steganography technique applied
GUI	UI/Main Window	Graphical user interface
Embed	Encode/Hide	Insert text discreetly
Decode	Reveal/convert to plaintext	Retrieve the original plaintext from an encoded document
Plaintext	User text/Decoded text	Specified text to be hidden within a document
Waterfall Model	Development model/waterfall development	“a sequential development process that flows like a waterfall through all phases of a project”[43]
Higher classes	Parent class	Code classes that are inherited from below
Method	Function/feature/Operation	An operation comprised of steps carried out within code
UML	Class diagram/UML diagram	Unified Modelling Language
Sample	Placeholder/Dummy/Test data	Data which is used for testing purposes in code
Knowledge base	KB/Databases/Pools	Pool of data which the program can pull from
LFA	Letter Frequency Analysis	Ordering letter by most common usage

(Fig. 2.1)Table of relevant terms

2.1 Methodology

Going into the development of this project the aims and objectives were previously clarified in this document and the *Initial document*. To fulfil these aims and objectives the project would need good direction and steganography would need to be heavily considered before making a time loss by investing into the wrong/redundant approach. Considering that this project was individual this gave it freedom to explore multiple concepts and a varied/diverse development methodology, generally sharing aspects of many models such as ‘*waterfall*’ and ‘*extreme*’, developing new features came out of necessity to fulfil aims in instances that this was not achievable exploring new methods and techniques took priority. Diagram of development progression:



(Fig. 2.2)Development/development relationship diagram of primary classes

The chosen approach was to use a series of whitespace techniques placed and non-format and function impactful portions of the code, also integrating security techniques to be more efficient letter frequency analysis was used to reduce space but also produce less noticeable patterns and footprints. In order to develop this vision three main classes were the focus of the project’s development methodology, these were the writer, reader, and user interface also developed in this order.

‘*Writer.py*’ would be the class responsible for producing the embedded file and so shared many similar requirements and interactions between classes. Following an improvised waterfall development considerations for requirements evolved along time. In example the input portion which needed to be encoded by the writer file was

originally sample text before being changed to a custom line and eventually user input once considerations for compatibility aspects of the program's scope were secured. This bridged the gap into design and made evolving development of features needed to secure an initial working version trickle down into other affected classes as can be seen in *Fig 2.2*. *Reader.py* was heavily influenced by the design of *Writer* as their functions are directly correlated, the back and forth between developing the initial version constituted much of the design portion. Full testing could only occur once both *Writer.py* and *Reader.py* were compatible, this is not the case for testing aspects which do not relate to embedding and decoding as they are more easily observed in comparison to reversing embedding by counting whitespace.

'*Reader.py*' responsible for reading what was produced by *Writer.py* was developed in tandem with *Writer.py* meaning many of the requirements from classes above it had to be passed down to be adapted to by *Reader.py*. The main requirement was to provide translation from a selected HTML stego file with the design and implementation of this feature being heavily inspired from code within *Writer.py* as conditions for embedding can be mimicked to produce the embedded letter. Implementation is passed down from *Reader.py* to ensure compatibility then the same trickle-down effect also applies to UI as outputs can change from this file and inputs can vary from *Writer.py*. To ensure that the outputs are in an appropriate format, the structure of the cover text is closely adhered to by each file, this also means that debugging is easier and the user is able to see noticeable differences when it comes to comparison.

'*MainWindow.py*' acting separately during development followed success of the other classes requirements but was permitted freedom when it came to design as many decisions were reliant upon readability and ease of use. In short, the primary requirement was to implement all functions within each file into the user interface involving having functions to display the embedded and decoded text while also having functions to sanitize and re-embed within the same window. Implementation was also solely dependent on the format outputted by the previous higher classes, so each implementation was a consideration from design of other classes. This usually

followed the process of using *QTdesigner* (see. *Sect 1.6 Tools, Environment and Resources*) to set out a visual outline for usability with function still in mind, then converting using *PyUIC* (*sect. 1.6*) once readable individual functions are tweaked to display output and input from higher classes as well as other 'keep-alive' features being adapted as it is a live GUI. Unlike the other two classes the GUI could effectively be tested with sample/placeholder data as the functionality does not lie within its own class.

After each class was able to be tested and updated in accordance to be compatible with each other a series of adaptations took place based off observation, good practice, and efficiency/complexity improvements. As shown by *Fig 2.2* each

adaptation was taken on by all classes as their functionality was intricately linked with few exceptions, following this a version would be assigned, design reconsidered, and testing repeated.

The two main pieces of data used consisted of the cover text and plaintext. The system needed to be able to handle a variety of each input, sample text was originally used to mimic a simple HTML document and a basic message to encode.

Plainhtml.html

```
<!Doctype html>
<html>
<body>

<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>

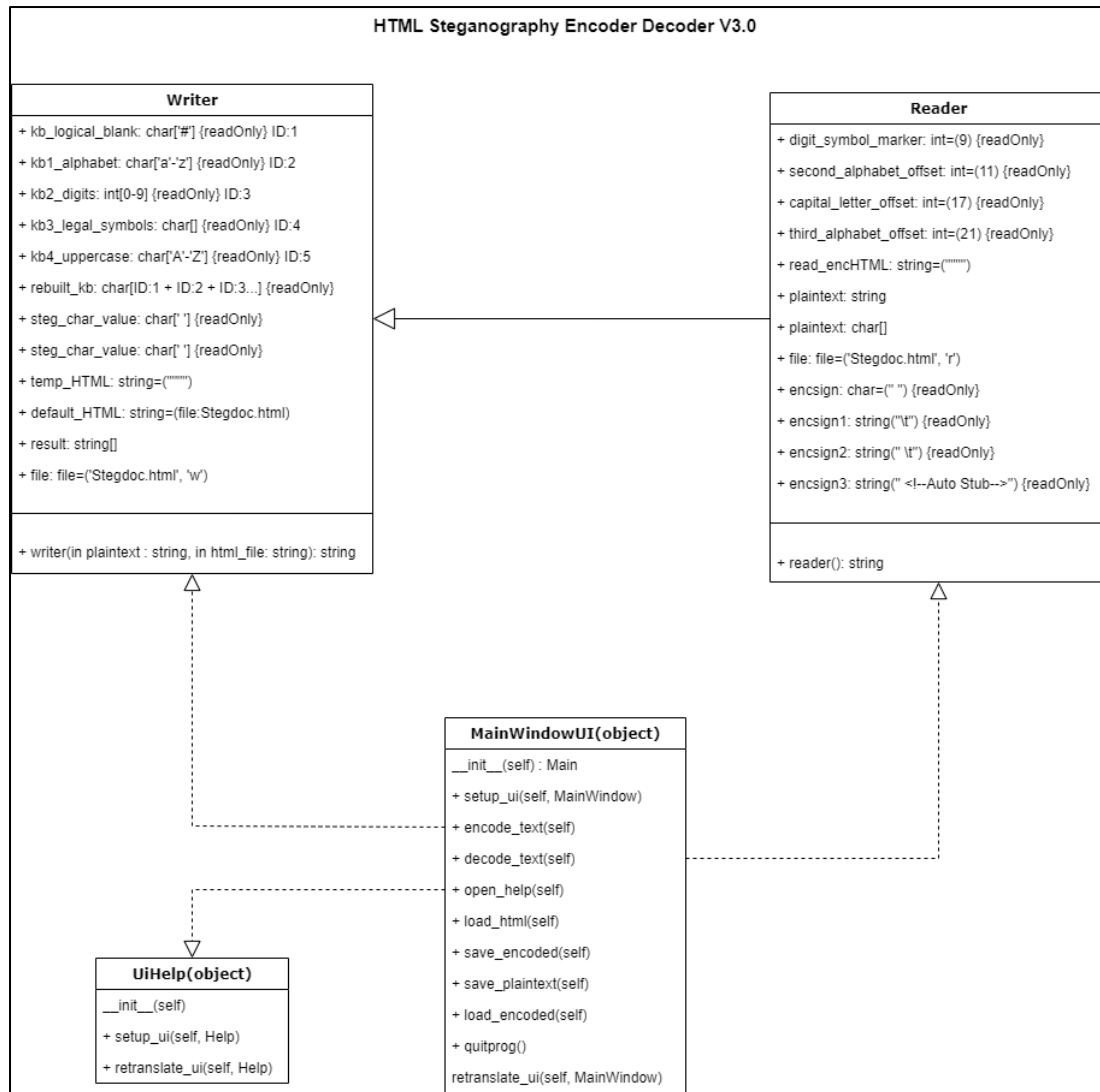
</body>
</html>
```

(Fig. 2.3) Sample text used as HTML text testing

These samples were later expanded to be more complex with the plaintext to encode becoming 'T3\$t-te?t' in later versions(2.0-3.0) and the sample HTML becoming closer to representing what you may find on an ordinary web page. By using this type of data, it was easy to setup robustness and security tests that advance in stages based upon complexity of the plaintext and cover text.

2.2 Program structure

This section will cover the operation of the program in a walkthrough manner explaining methods and usage from a review and user standpoint, this section should be referred to when considering the techniques used. Further details into the thought process and considerations behind specific design and logic decisions are covered in *Sect. 2.3 Algorithm Design*



(Fig. 2.4) Class UML Diagram of version 3.0

Based upon the above UML diagram the project uses the user interface to implement each of the functions including a small new window for help information, the general operation will be explained in an advanced user end manner.

Firstly when the program is started the *MainWindow* is run as 'main' to initialize all other objects which are needed to display a live GUI including: Buttons, Fields, output, titles, tip text, and the window. This is replicated to show the *UIHelp* window and applied to a button press through the *MainWindow* function 'open_help(self).'

To encode a simple HTML document the user has multiple options a HTML document can either be loaded via an open file button or with an empty field the box will be filled by sample text as an example of how the encoding will look. Once the user has entered a document and the plaintext they wish to encode, pressing the encode button will pass both of these attributes to `writer(plaintext, html_file)`.

Writer

Within the method the algorithm first strips the given plaintext string for every character and converts to a list of char with the line

`plain_list = list(plaintext.strip(" "))` then moving onto the cover text a similar operation is followed instead now for each new line if there exists a ‘>’ as this denotes a tag.

Next by using dual iteration the algorithm can cycle through the longest using ‘`itertools.zip_longest`’ of either the plaintext list or the HTML list, if this is not possible or exceeded the control flow uses ‘`fillvalue=#`’ to replace invalid lines also helping to catch errors.

There are a series of conditions for encoding each character which is legal, these legality rules come from a series of ‘*kb*’s or knowledge bases. Simplified the rules are as follows

If the line is valid, not uppercase and the first letter of the plaintext that needs to be encoded has the index between 1 to 9 in LFA order, then combine the needed value based on index amount in spaces.

If the first conditions are met but a plaintext letter with the index 10 to 18 produce this letter by using the index amount in tabulation.

If the first conditions are met but a plaintext letter with the index 19 to 26 produce this letter by using the index amount in space and tabulation.

If the first conditions are met but a plaintext Digit or Symbol with the index 28 to 54(as # is a reserved symbol) produce this letter by using the index amount in spaces greater than 9.

If the first conditions are met but the plaintext letter is a Capital produce this letter by using a comment and the index amount in spaces.

see. Appendix B for full code

Once the length of the longest attribute has been exhausted the algorithm will re-compile the string with the appended additions to produce a stego doc by writing the changes to the file.

Reader

In order to decode a stego file the user must press the decode button while there is encoded text within the encoded text box, the program requires no further input to convert back to plaintext. The algorithm in reader starts by reading the stego file and splits it for each new line, the program keeps a collection of variables that are associated with each type of encoding from the writer labelled as 'encsign' 0-3.

Again, using iteration the algorithm searches for these labelled signs at particular portions of a line using a key offset dependant on what range the character is.

The last third of the alphabet and digits/symbols can be detected using *splitSteg[s].find(encsign2)* automatically as they aren't a substring of an individual encsign. The reader is placed one space ahead of each closing tag '>', the following conditions are then followed to translate back into a plaintext character:

If the algorithm finds a string matching " \t" then take the right stripped difference of the rest of the line(len(stegdoline)-len(encsign2)) leaving the number of encoded append this number using the offset of the third alphabet along the length of the combined knowledge base.

else if the line at one position ahead of the html line contains a space and does not include tabulation or comment. Append the remainder of the line based upon number of spaces applied to the knowledge base.

if the number of spaces exceeds 9 however apply this length to the digit symbol knowledge base.

else if the line at one position ahead of the html line contains tabulation and does not include space or comment. Append the remainder of the line based upon number of spaces applied to the knowledge base.

If the algorithm finds a string matching " <!--Auto Stub-->" then take the right stripped difference of the rest of the line(len(stegdoline)-len(encsign3)) leaving the number of encoded append this number using the offset of the third alphabet along the length of the combined knowledge base.

see. Appendix B for full code

After this process the result is joined as 'plaintext' when it is returned.

2.3 Algorithm Design

This section aims to document the precise workings and decisions that account for the operation of the algorithm, many of these choices were a result of investigation of the previous sources or use of resources to perform tasks in a more optimal manner. Going into this project from a programming perspective the requirements outlined within the *initial document* would be fulfilled by using a null space approach but as previous sources within the literature review have evidenced this project would need to innovate further to introduce an ‘*aspect of security*.’ The first of several design decisions would come off the back of this motivation

Security

The most significant design choice that was made to produce an aspect of security was the letter ordering scheme. It is common among many steganography schemes that represent a number in a bitwise fashion to apply this number directly to an encoding table such as ASCII[34] or UNICODE[45]. Should the secrecy be compromised numbers like 65 and 97 corresponding to ASCII ‘A’ and ‘a’ will be easily identified and even decoded as there are only two permutations of each symbol 1 or 0. This project has found an alternative means to maintain security using a lesser-known scheme and encoding per line with new identifiers. This is implemented by having several knowledge bases to split the text using letter frequency analysis order, each split knowledge base also has an ‘*encsign*’ so that it can be identified by the reader. Letter frequency analysis order was chosen to reduce space being used at the end of a line, this improves secrecy but also security as each third of the alphabet among other common symbols will appear with subtle variation to the prefix confusing attempts to discern at what region the split occurs and under what sign.

Writer.py

```
kb_logical_blank = ['#'] # For lines that fail to encode

kb1_alphabet = ['e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'l',
'd', 'c', 'u', 'm', 'w', 'f', 'g', 'y', 'p', 'b', 'v', 'k', 'j',
'q', 'x', 'z'] # Letter Frequency analysis order reduces size

kb2_digits = list(string.digits) # 0-9

kb3_legal_symbols = [' ', '!', '$', '%', '&', '*', '(', ')', ':',
';', '-', '+', '=', '?', '@', '.']

kb4_uppercase = ['E', 'T', 'A', 'O', 'I', 'N', 'S', 'H', 'R', 'L',
'D', 'C', 'U', 'M', 'W', 'F', 'G', 'Y', 'P', 'B', 'V', 'K', 'J',
'Q', 'X', 'Z'] # Letter Frequency analysis order
```

(Fig. 2.5)Writer class databases ordered in LFA

1st set split (e to r) with sign ' '
2nd set split (l to y) with sign '\t'
3rd set split (p to z) with sign '\t'

This design decision did further complicate the algorithm, but this sacrifice was worthwhile to produce a unique algorithm that would also complicate things for potential attackers. The symbols and alphabet were chosen specifically to allow access to the most common characters except in the case of capital letters as they are optional and may not occur often. While having minimal restriction, this restriction initially came around due to processing time but is also a factor when splitting and re-ordering values.

Secrecy

Admittedly secrecy is one of the more important aspects of this algorithm hence the choice to pursue null embedding schemes to eliminate the appearance of text.

Fortunately in HTML only text encompassed by tags is read because it is a mark-up language[44] this allowed the algorithm to be designed in a manner which maintains secrecy and does not interfere with operation of the tags themselves.

The way that characters are embedded (partially explained in *security*) append to the end of the line, this was a deliberate design decision that was prototyped and iterated from alpha versions where the embedding would be embedded at the start of the line. Embedding at the start of the line encountered problems and displayed less secrecy, despite looking like spacing and indentation this technique would be suspicious when applied to HTML as the language is not sensitive to indentation, not only this if the algorithm was to use indentation as grouping the groups would appear non-sensical when a variety of characters was encoded.

alpha writer.py

```
if f.endswith('>'): #catches normal line cases
    temp = rebuilt_kb.index(b)
    f = (('a' * temp) + f)
    result.append(f)
    # print ((rebuilt_kb.index(u) - 26))
    # print (f)
```

(Fig. 2.6)Early alpha class using pre-line encoding

Using end of line embedding was a preferred alternative that offered advantages over the scheme presented in Fig. 2.6 primarily it would produce stego text without visible irregular indentation, the exception to this is comments however many of these cases are often dismissed as variety formatting. By choosing to encode at the end of the line the algorithm exhibits greater compatibility with general HTML files found online as space will exist at the end of each line and will not disrupt

indentation if any is present. The encoding signs used were also a consideration when designing for secrecy, these also had to be closely tied with security as this would be the first line of defence should an attacker seek to obscure or decode the embedding. Each are designed to be blank but unique with the exception of embedding capital letters which are optional. This allows the user to directly choose the outcome of the stego doc based upon letters used, for instance; A user may opt to use an 'i' instead of an 'l' so that the split can be maintained in the first third of the LFA alphabet, or they may opt to misspell a word to add an additional obfuscated layer which can be further enhanced by choosing which sign to use.

Robustness

Robustness refers to the integrity of the stego doc, if transferred to the wrong person steganography techniques which are robust will be resilient to attacks such as compression as described by *Neil F. Johnson, Zoran Duric, and Sushil Jaiodia* in *Information Hiding: Steganography and Watermarking Attacks and Countermeasures*[25]. Unfortunately, by nature the weighting of robustness is heavily influenced by the secrecy of an algorithm, a general statement could be made as; greater complexity in secrecy algorithms create large surfaces for attacks against robustness. This projects algorithm has some mechanisms to deal with disruptive techniques when attacked, primarily text preservation which is applied in all fields which can be changed by the user.

Firstly temporary variables are used and checked so that the algorithm is able to fall back on original input if new input fails to verify.

Writer.py

```
temp_HTML = "" # preserve original input

default_HTML = ""
{TEST_DATA}""

result = []

plain_list = list(plaintext.strip(" ")) # convert to list

# may be a requirement that the sample is longer to fully
encode

# writer will default to default_HTML if there is no valid
text present to encode
if html_file == "" or html_file.find('>') == -1:
    file.write(default_HTML)
else:
    file.write(html_file)

file.close()
```

(Fig. 2.7) Mechanism for file preservation/handling

A similar mechanism is also present in *Reader.py* when loading stego files. By using

temporary variables, the algorithm can ensure that it is reading original input rather than regenerated stego text from a previous runtime, this aspect also increasing usability by reducing potential for user error.

Also aiding in usability but providing benefit to robustness, the user interface of the algorithm prevents input into the intended outputs of the algorithm; these fields being the stego doc output and the plaintext output. This is implemented by marking specific fields as read only within the UI instance.

MainWindowUI.py

```
self.Encoded_sample = QtWidgets.QTextEdit(self.centralwidget)
self.Encoded_sample.setGeometry(QtCore.QRect(500, 100, 211, 411))
self.Encoded_sample.setAutoFillBackground(False)
self.Encoded_sample.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
self.Encoded_sample.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)

self.Encoded_sample.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustToContents)
self.Encoded_sample.setLineWrapMode(QtWidgets.QTextEdit.NoWrap)
self.Encoded_sample.setReadOnly(True)
self.Encoded_sample.setMarkdown("")
self.Encoded_sample.setObjectName("Encoded_sample")
```

(Fig. 2.8) Encoded output field parameters

By restricting these fields to read only we reduce the risk of allowing legitimate users or other parties from interfering with the outcomes of each method. When it comes to reading the file, this is important as discrepancies to otherwise valid embedding can cause plaintext to be displayed incorrectly if this was not the user's intention.

2.4 Findings and Evaluation

This section is dedicated to expressing the most important transferable information that is spread across multiple versions of the algorithm. Other factors supported by research and analysis will be brought into the findings and conclusion to better substantiate what constitutes useful information that can be applied elsewhere in the field or for an individual to pursue a similar project.

(Each of these versions can be found at: <https://github.com/Ju5t-j1m/Y4project>)

Version 1

The first version of the chosen algorithm also containing three sub versions focused on embedding using two knowledge bases, producing schemes which were far longer and exhibited worse secrecy than later approaches due to the need to encode sequentially using the same knowledge base. The version progressed to account for a greater range of input using the test string 'at4st-text' each version tested the output was valid before integrating further features.

Evaluating this version of the algorithm using the metrics Security, capacity, and imperceptibility as described in *A Review on Text Steganography Techniques* by M.A.M, R.S, Z.S, and M.K.H[22]. Version one displays a decent level of imperceptibility due to a null approach but ultimately suffers when under further scrutiny, simply by highlighting the text a large embedded message will appear after each tag in the form of spaces, moreover the size of each embedded character will be greater depending on position meaning in some instances the file will expand horizontally and allow for scrolling, this is not an issue in all word processors but those which adapt alongside line length will appear suspicious when a steg file is read. Statistical approaches will also produce outliers when compared to a normally formatted HTML file, the quantity of space characters will be noticeably greater making it easy to detect embedded text. Despite this security risk the algorithm is somewhat resistant to disruptive techniques aimed at the start of each line, alongside disruptive/compression techniques that are applied through the program itself as the program sanitises the HTML doc ensuring better compatibility. Capacity was this version's greatest weakness, with the potential to ruin embedding by feeding onto the next line should a character with a high cardinality be encoded only a very limited selection of characters could be chosen, on top of the limits already imposed by the groups used.

Version 2

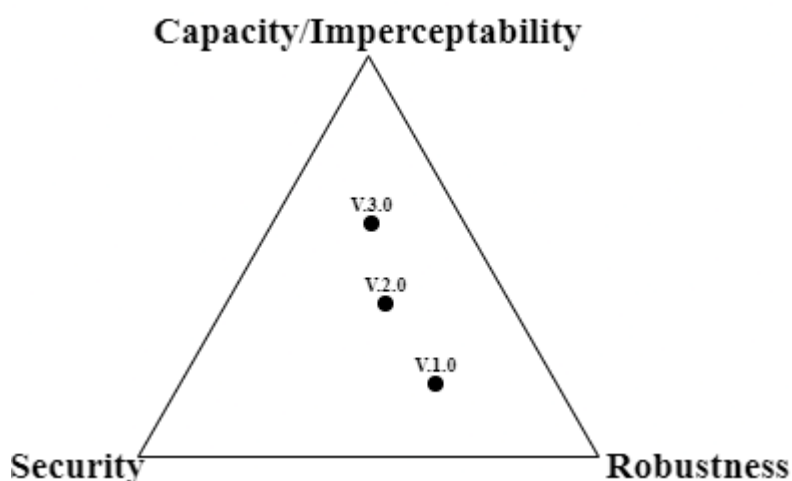
The second full version of the algorithm saw the implementation of the knowledge base grouping scheme in attempt to solve previous capacity issues but also improve imperceptibility. Splitting the knowledge base of characters gives greater security over a continuous list as the offsets/keys will have to be identified to retrieve the algorithms method, version two was also the start of the development for alternative signs to make larger knowledge bases more distinct; a factor which would later provide another aspect of security. Version two displayed similar robustness to version one but is more complex in order to create a more secure and imperceivable encoding style, meaning techniques that involve obfuscating data or compressing data would be more effective against this version as each character encoded now has more specific information related to it. Each knowledge base containing nine characters improved imperceptibility by splitting the total length of each embedded character avoiding the risk of scrolling and reducing risk of basic text analysis techniques to fall closer to what is normal from a HTML file. The average line length as a result of the grouping of three knowledge bases saw improvements over the previous singular sequential knowledge base, making the max line length the HTML text combined with a maximum of ten spaces after. This improvement offered greater capacity as the length of a line was no longer at risk of consuming two lines which would reduce the capacity factor by $\frac{1}{2}$.

Version 3

The third version of the algorithm contains all features necessary for a definitive version, these features are perfected alongside a user interface. The internal goal for a

definitive version of this project was to have an algorithm which operates in reasonable time, produces an End-of-Line Space cipher consistently and accurately, and can be operated by most users displaying some robustness and aspects of security to protect the previous internal goals and the user. Version three provided more security over the previous two versions with the full introduction of ‘*encsigns*’ and a complete list of knowledge bases by which the embedding was split, many features intended to develop greater imperceptibility also became closely tied to security of the program. Another example in version three would be the utilization of LFA to shuffle letters, it’s worth noting that this facet of the program benefited all aspects aside from robustness. By utilizing LFA order and dividing each knowledge base this program improves upon imperceptibility and capacity by reducing the number of characters used per line based upon English language usage. The knowledge bases are also split accordingly so that more frequently used letters appear with simpler shorter ‘*ensign*’s, also making intrusive techniques harder to perform. Robustness has a marginal improvement over version two by introducing more checks to verify the integrity of the original HTML file and detect any previous encoding or leftovers.

Using the triangular model of Capacity/Imperceptibility, Security, and Robustness proposed in *A Review on Text Steganography Techniques* by M.A.M, R.S, Z.S, and M.K.H[22]. Each version can be briefly classified to give a better understanding of the trend of priority and improvement over the development of the algorithm. As is the purpose of a triangular graph a singular algorithm will not be able to fulfil fully all aspects important to a steganography algorithm. Security will limit robustness, robustness will limit capacity and imperceptibility,



this will then in turn limit security. Given these restrictions this is where each algorithm plots in respect to each other:

(Fig. 2.9) Three versions of the algorithm plotted on graph described in[22]

Through developing these three versions information can be taken from the methodology to aid in further or alternative project developments, these findings

were the result of progressive and iterative development as many of the features were added to accommodate aspects of the algorithm which did not suitably fulfil objectives. Null embedding still creates noise, simply because the embedding is not visible to the eye does not mean that it is unimportant to optimise and make considered decisions on method used, null embedding if not used properly can be just as obvious as substitution. As learnt from research it is important to include security-based aspects in steganography algorithms, this will afford more security to the algorithm and should it be applied to serious or non-serious use cases a security feature will see benefit in either. Take into account anomalous lines, each algorithm can choose to deal with the differently, but they should be expected and handled for example version three checks the documents tags before writing, failing this it handles the error.

Overall the development of this algorithm and subsequent program was successful, this programs success can be attributed to the iterative development making it possible to meet goals and adapt a program freely with little constraint, although this methodology traditionally lacks direction it aided in innovating features for the program. Many of these features were inspired by similar projects which were researched within the *Literature Review*[10-11][16-18][25] and allowed cross comparison and a continued style of research which would aid in development. The developed algorithm does display shortcomings within robustness and security(see. 4.1 *Conclusion*) however each version improved these aspects from the initial state of the algorithm and are successful in communicating a null embedding scheme. The algorithm can still further be improved(see. 4.2 *Future Work*) and continued iteration is carried out to publish new versions to the repository[41].

End of Chapter 2

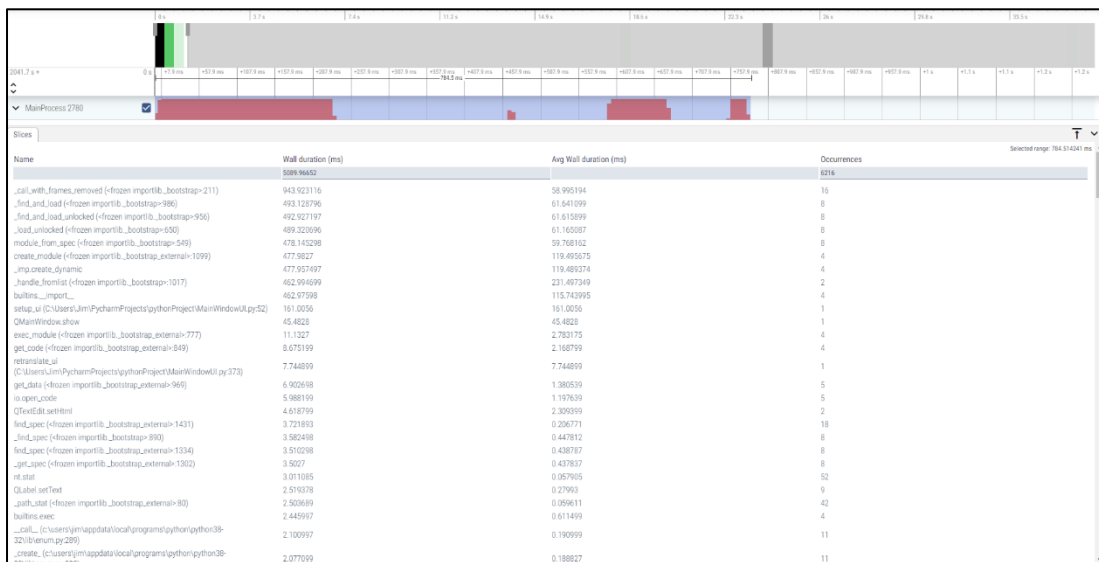
Chapter 3

Analysis

This section will go into detail on the methods, comparison, findings, and results of close analysis of the final developed algorithm for this project in isolation evaluating it in terms of efficiency, effectiveness, and more broadly its wider metrics that contribute to its value as a steganography algorithm. Alongside this, other algorithms researched within the *Literature Review* will be compared on the same terms to determine the value of each to a project similar to this one and steganography as a field. Following this a brief discussion on the value on the different approaches when compared based upon metrics, these discussions will inform the findings of the work as all previous sections will be concluded within *Chapter 4 Conclusion and Future Work*.

3.1 Efficiency and Effectiveness

From a high level/user perspective the program runs and performs all methods within the ui in acceptable time defined as between ‘0.1 second for directly manipulating objects’ and ‘1 second for freely navigating’ by Jakob Nielsen[47]. These requirements are applicable to this user interface as the user has direct influence over the actions and input used. The average execution time of the real time UI booting and ready for input is 757.9ms as profiled by VizTracer(see. 1.6 Tools, Environment, and Resources) this is suitable booting time and will support a number of operations performed within the main window.



(Fig. 3.1) VizTracer profiling view of UI booting process in appropriate time cost[46]

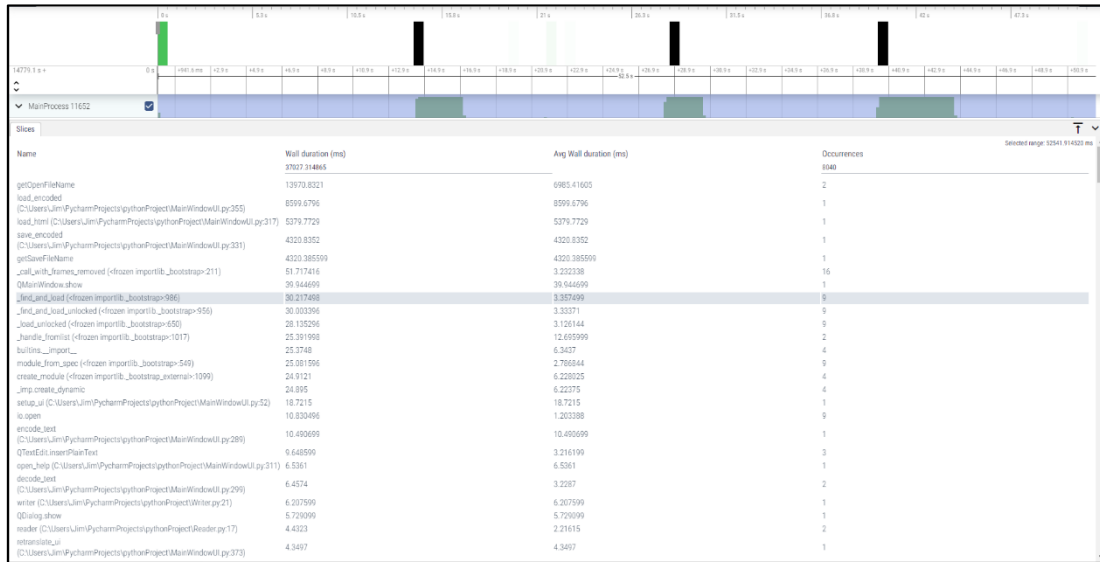
3.1.1 Program Efficiency

Program efficiency concerns the functions of the program in comparison to the time cost, memory cost and processing cost. The program can be measured in each of these areas to gain a better understanding of program efficiency which can be used to aid in evaluation comparatively and overall. Metrics such as time cost and processing cost should be closely tied with some dependency on threading and potential errors within the program such as a memory leak. Since this program does not operate in an online manner consideration for networking efficiency and or bottlenecking due to networking can be disregarded.

Time cost

The main time cost from the program comes from booting of the UI which takes considerably longer at *84.1ms* than other functions of the program, this may be classified as permissible as the setup is for essential components such as the *formlist handler* responsible for managing objects at *80.5ms* across the duration of bootstrapping. Also, a live UI is not taken into consideration when executing each function within it. Along with this *1 second* or below is acceptable time even with the bootstrapping of the program and UI.

As for core functionality the program is primarily operating off the *Writer.py* class(see. 2.2 Program structure) taking *6ms* the main control flow of this class relies off the *zip_longest* function within python which runs off $O(N)$ time complexity, in this instances selection occurs between the length of the plaintext(in characters) or the *html_file*(in lines). This means that $O(N)$ is changed depending on longest length $O(N(\text{html_file} (>)(<) \text{plaintext}))$. Outside of the main iteration control flow the program then uses five distinct selections to filter the text to encode this does not concern $O(N)$ as the selection involves constant factor(k), this is likely the largest complexity of the algorithm as is reflected within the profiling results with 45 occurrences of *.isupper* despite this constant inefficiency with a small data selection and complex legal plaintext execution of *8ms* is very satisfactory for a UI application. *Reader.py* shows similar time complexity of $O(N)$ where N is the length of the split encoded text, constant factors within this iteration are far simpler to evaluate resulting in less time cost from selection. This is reflected by the profiler with execution of decoding only taking *2ms* with 20 calls to *.find*. Calls to *save* or *open* a file are time dependant on the user temporarily freezing the program and call to windows functions outside of the programs control, for this reason time cost due to these waits is not considered for full evaluation. Opening the help window runs similar calls to bootstrapping the main window however executes in far less time only consuming *3.4ms* less than that of encoding a file.



(Fig. 3.2) VizTracer profiling view of full process in appropriate time cost[46]

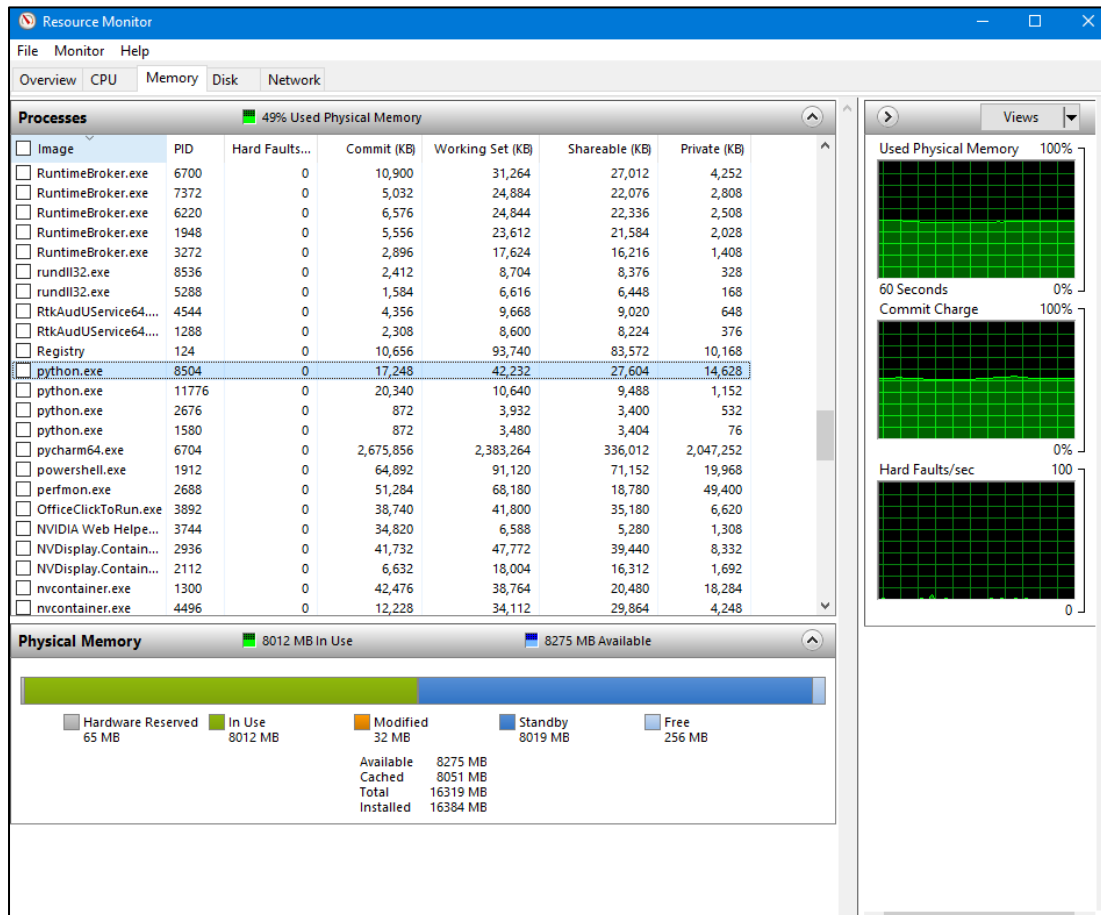
<i>Data used</i>	<i>Embedding cost(ms)</i>	<i>Decoding cost(ms)</i>	<i>Total program cost(ms)</i>	<i>%Difference</i>
Simple Data (Basic plaintext & HTML)	7.1	1.9	93	N/A
Average Data (Alpha numeric plaintext & webpage HTML)(2.13 KB)	9.2	4.5	97.9	^+5%
Complex Data (All legal plaintext & complex webpage HTML)(146KB)	223	62	369.1	^+296.8%

(Fig. 3.3) Table of time cost (Booting/Embedding/Decoding) of three html samples

Memory cost

Memory cost is also a consideration for this program when working with large pieces of data, HTML files can become increasingly large with Google indexing up to *15.7MB* pages[48]. Though the program is not built for pages of this excessive size it is still important to consider data input, output as well as how the program sits in memory when idle and when processing. For this we can use tools available on the windows system such as process explorer and Resource monitor, by tracking the memory usage the cost can be calculated across each function of the program. While idle the program uses *43,860KB* or *43MB* much of this memory is used to display UI elements as many of the features need to be translated and displayed using

an external library. Simple operations using the interface consume an additional *100KB* such as *Help* and other more complex actions such as ‘*Encode*’ and ‘*Decode*’ consume *400KB* respectively however much of this memory is reserved for future actions as *commit* memory virtual memory ready to be processed. An example of this usage is when memory spikes up *3000-4000KB* upon performing windows functions, this memory is then reserved for later use in the program but remains shareable.



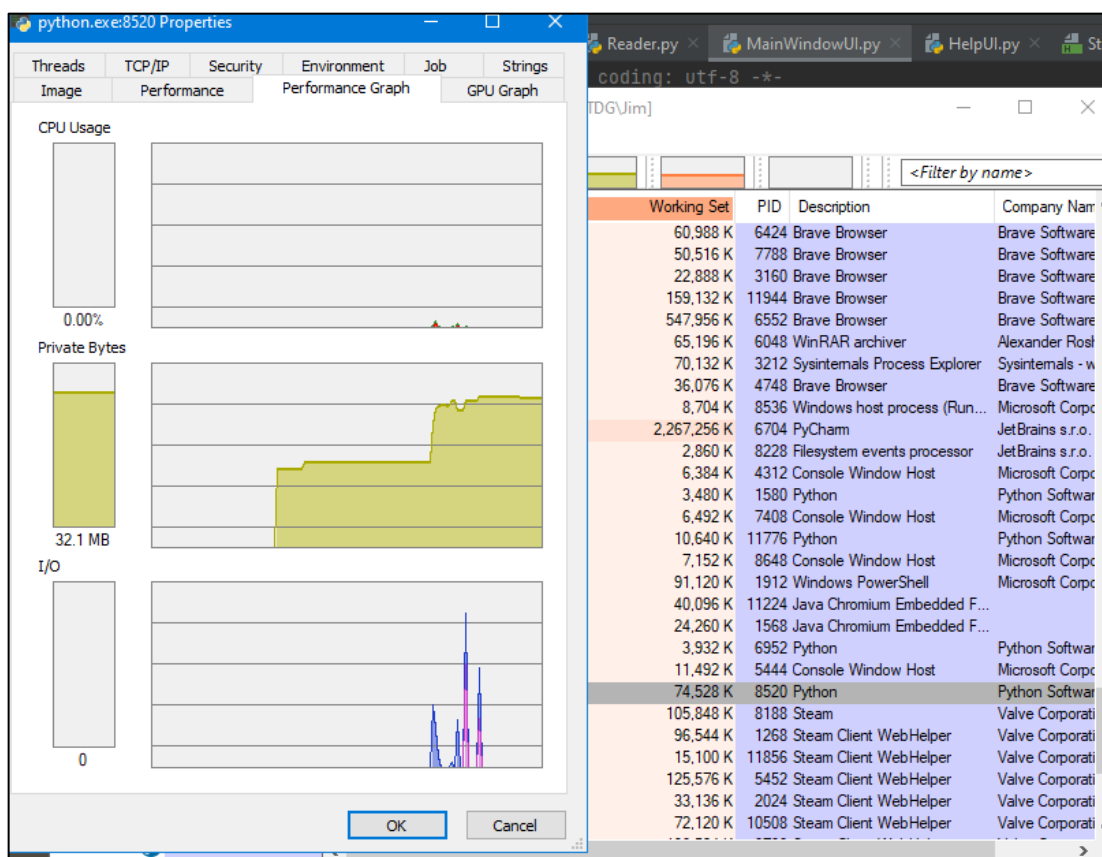
(Fig. 3.4)Resources monitor tracking the memory of the python process during idle

<i>Data used</i>	<i>Embedding cost(KB)</i>	<i>Decoding cost(KB)</i>	<i>Total program cost(KB)</i>	<i>%Difference</i>
Simple Data (Basic plaintext & HTML)	+400	+300	44,260	N/A
Average Data (Alpha numeric plaintext & webpage HTML)(2.13 KB)	+560	+452	44,420	^+0.3%
Complex Data (All legal plaintext & complex webpage HTML)(146KB)	+2044	+1130	45904	^+3.5%

(Fig. 3.5)Table of memory cost (Embedding/Decoding) of three html samples

Processing cost

Processing cost is closely related to the other two types of cost but is also very hardware dependent, this test machine runs off an *i5-9600k @ 3.7GHz* with *16GB of GDDR4 RAM* making processing time much faster and with the incorporation of some technologies will also perform more efficiently in comparison to other machine. Processing cost overall for the standalone program is inexpensive only utilizing around 3.2% of the CPU when loading a costly file. While idle this is more negligible not even appearing to have any passive signature on the CPU graph displayed by the process explorer. This correlates with both other areas of efficiency and speaks to the size of the program, essentially being a series of intricate scripts behind a user interface to ensure ease of use.



(Fig. 3.6) Process explorer view used to capture CPU information of the program

Overall the time, memory and processing cost all fall within the bounds of 'acceptable' however this does not mean that it is without flaws, one particular area that would be best to improve for an increase in efficiency would be the writer evaluation. By creating better evaluations, the program can reach a more appropriate *k* value for selection process flow, this is an apparent issue due to the number of specific evaluations that are required for each line. Most user machines should have little difficulty running this program with suitable performance, the program is also small enough to run on said machines. These results will be kept in mind when considering other approaches and the overall summary and place within the field.

3.1.2 Program Efficacy

The performance of a given program may be acceptable, but this does not account for the intended operation of the program, this is where efficacy is involved within the evaluation of the program. Can the program operate in the intended manner and achieve expected results? To a degree this will involve performance, but this section is more concerned with the input/output and expected operation of each function and for this reason will go through each aspect of this program.

Input

Firstly user input is a key component of this program, although restricted to a set of '*legal characters*' the user can encode a message composed of lowercase, uppercase, numbers, and a variety of general symbols. The program validates this via regex to avoid errors caused during encoding, the program also ensures that the number of lines to be encoded to is valid by setting the max length equal to that of the default sample or a loaded one. The input is passed to the Writer class in a compatible format and processed according to the classes' rules. The input acts accordingly to the rules defined to the field but also more broadly respects the rules of the programs execution taking into account the permitted length and characters.

Output

Similar to the input the output is reliant on the '*Decode*' function in the same manner that input is reliant on '*Encode*.' The results of decoding can be if the number of lines present in a HTML file, the field is able to display this many characters but will need to be copied or saved via the adjoined button in extreme instances for precision. Because of this simple functionality the output performs as intended by the rest of the program it is tied to.

Encode

Due to the diverse nature of HTML files it is difficult to ensure a full range of compatibility especially since HTML as a language still being developed upon. A potential limit to this functionality is max line length of a given file, some exceedingly long HTML files also include reference to an XML or CSS file, these tags can be longer than the rest of the tags if appended through saving and not re-formatted. There always will exist an instance in which the data cannot be encoded to as there will always be a predefined line length, this is a disadvantage of end of line encoding as too much space will be consumed to encode anything. The encode button takes its data from both the HTML and plaintext field then passing to *writer.py* format of the file is checked within this class and will return an error if it cannot be encoded to. With exceptions to extraordinary files the Encode function produces predictable and measurable results(see. 3.1.1 Program Efficiency)

Decode

Decode is alike Encode by its operation but only has one input for the encoded text, the program ensures greater compatibility by setting the encoded field as read only to avoid accidental keystrokes or incorrect adjustments being made to the file. Due to coding similarities, if the file can be written by the writer it can be read by the reader as the cross compatibility is high. However, this means that the reader also suffers from similar drawbacks such as the length issue, decoding is also at risk of generating the wrong plaintext in rare instances, this is a serious fault for the user if the original input is not known as a concealed message could be misinterpreted. Decode maintains the same functionality as Encode and rarely produces incorrect decoded results, using several examples from numerous online webpages has verified this and direct comparison to the efficiency data also contributes.

Help

Help is intended as a user guide and is an addition to the interface not serving a programming function aside from a new window to display more information. The dialog that is opened does not interfere with the functionality of the rest of the program and normal operation can resume once the user is comfortable with the help displayed.(see. *3.1.3 Program Interface*)

Save/Load

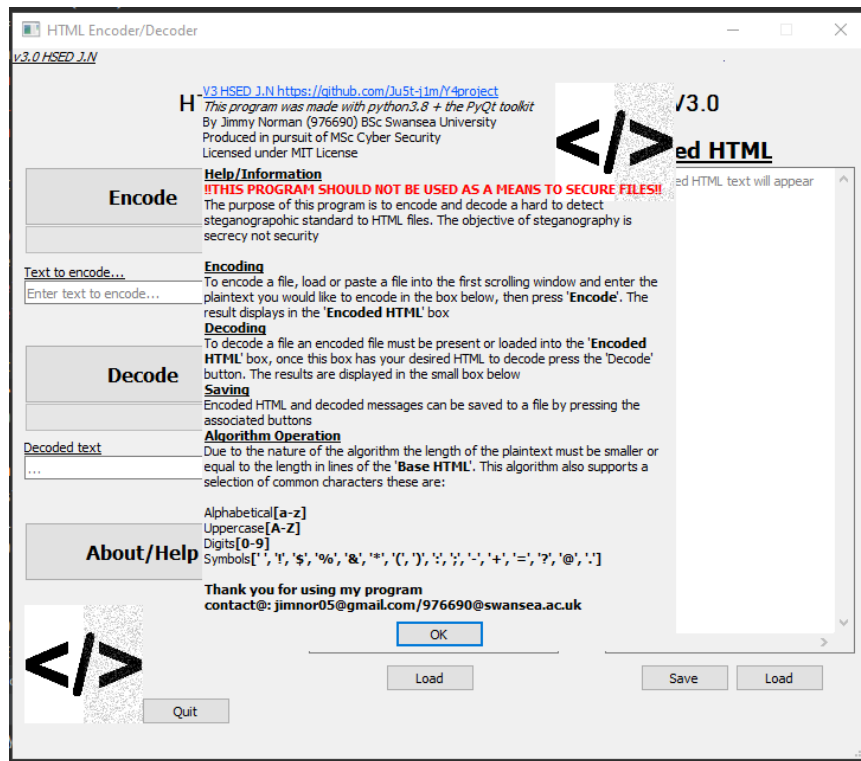
Save and load functions are handled by a Windows call after the button is pressed, parameters to set accepted file types are also passed during this call, the program specifically focusing on `‘.html’` and `‘.txt’` to load a file from text and will also save in similar formats. Since the program is portable the option to create a custom directory is given and will be recalled by windows for the next save. Within the code the step doc is tracked for debugging purposes but also to clear and contain each new encoding to create greater robustness.

Quit

Quit is a simple quality of life feature that ensures the program exits safely using exit code `‘0’`. This is important as the program handles and manipulates files so closing properly will ensure the preservation of data; the function performs this correctly.

3.1.3 Program Interface

Developing the algorithm into a simple user interface has provided benefits to a range of users and aided in the automation of encoding and decoding documents. Some aspects of the interface help to convey intention behind the program and can be used to prompt any user to navigate and interface with the GUI in a way which is goal achieving. Within the program a dominant component of this is the help interface.



(Fig. 3.7) Help interface to aid users in interacting with the program

The design of the interface kept basic principles of UX and UI such as ‘Reducing cognitive load’, ‘Consistency’, and ‘Visual structure’[49] at the core to reduce friction when operating a rather complex algorithm under the hood. Users will be informed of actions results by placeholder text as is present in the text fields before entry and tooltips available when the mouse lays idle over an object. The aim of these features was to give users information in a digestible manner if the help screen was not the preferred way of learning the features of an algorithm. Saving progress in files also helps maintain a sense of ease to the user as features can be more readily experimented with, without a sense of diminishing progress. Features such as a copy function were considered for fields which can become excessively long however were excluded to avoid clutter as save features are already present and will result in making longer outputs far easier to read. Labels also help to illustrate the purpose of each section especially for each larger input/output fields, these are the most vital components so warranted a larger label to highlight their core function.

3.2 Comparative Analysis

It is important that the outcome of the project is compared closely to similar techniques and available standards online, placing the algorithm among other similar techniques can be a good indication of efficiency and success overall should the algorithm be adopted. This section will use three comparisons to gain a better understanding of how the algorithm compares to the rest in the field.

An overview of text steganography R.K, P.T, M.B[10]

This source evaluates numerous types of steganography by providing a table of capacity in relation to line and word efficiency, 15 techniques are graphed one of which ‘*End-of-Line Spacing*’ sharing very close similarities and technique to the developed technique. The source describes the technique as:

“This method encodes the secret message by inserting white space or tabs at the end of each line (refer Fig. 5). Since nobody cares about the extra white space at the end of a line, this method might not create any suspicion to the third party. This method can encode much amount of data and uses two spaces to encode one bit (either “0” or “1”), four spaces to encode 2 bits (00, 01, 10, 11), and so on.”[10]

The key difference is how characters are defined, as explained within(sect. 2.3 *Algorithm Design*) the algorithm assigns by character in an efficient order also bearing similarities to the mentioned ‘*Character Marking*’. This source describes the ‘*End-of-Line Spacing*’ technique as ‘*Bit-Level (2-Bits)*’(see. Fig 1.3) with a capacity of ‘*2-bits/60-cc*’ with ‘*cc*’ being the average number of characters per line. This aligns with the 2-bit pattern as each blank will define a binary digit so this can be translated per line. If this algorithm were to be analysed and placed in a similar table, the type of embedding would be defined as ‘*mixed (1-Bit)*’ since it is a combination of character marking which is normally a 1-Char technique and *End-of-Line Spacing*. The algorithm is 1-Char as each line encodes using a series of spaces corresponding to a character, this could technically produce greater efficiency if chosen words are known such as: ‘*and*’, ‘*a*’, ‘*is*’, these words could be encoded using specific blanks creating a new mapping efficiency making the capacity variable. The developed algorithm shows the same/equivalent efficiency as *End-of-Line Spacing* being 2-bits or 1-character per line (where average is 60 characters). In context of HTML the capacity is subject to change as it does not abide by rules of English therefore it is hard to make an estimate on average line length, using an example the default HTML average length is 19 making the capacity *1-Char|Word/19cc*.

A Novel Text Steganography Technique Based on Html Documents by M.G[16]

Mohit Garg's unique steganographic technique provides a useful comparison for this project as it provides a HTML example of a binary 2-bit algorithm. The algorithms share similarities in their concern for HTML tags, where Mohit Garg's algorithm is concerned about the order of tagged attributes the developed algorithm is concerned with the ending tag of the work. The source describes the algorithm as:

"The proposed technique uses the html tags and their attributes to hide the secret message. It is based on the fact that the ordering of the attributes in the html tags has no impact on the appearance of the document. This ordering can be used to hide the secret messages efficiently."[16]

This technique is also similar in that it aims to maintain the appearance of the original document; however, it could be argued that a change in attribute order can break convention and is more noticeable than blank space. Garg's algorithm aims to embed 1-character/tag by focusing the embedding to tags this could positively or negatively impact capacity dependant on the type of HTML document as some tags do not require or operate without specific attributes. Using the developed algorithm can provide more general embedding as each line regardless of attributes can be embedded to. Using Garg's algorithm, it is also possible to embed longer strings of binary as some tags like '' and '<iframe>' commonly contain many more attributes allowing the algorithm to embed a diverse range of characters. To classify this algorithm with regards to the evaluation metric used by[10] the algorithm is of type 2-bit mixed with a capacity of 1-char/tag, this places it close to the developed algorithm in terms of overall efficiency but less visually secretive.

A compression-based text steganography method by E.S and H.I[26]

More sophisticated techniques involving compression and other greater forms of security partial steganography have seen an increase in popularity. This previously discussed aspect of security can be incorporated in many ways such as the developed algorithms character splitting, this source implements a technique that uses compression by iterating over the 'Lempel-Ziv-Welch' scheme by using stego keys to improve the bottle neck of capacity exhibited by its predecessor. As described by Esra Satir and Hakan Isik:

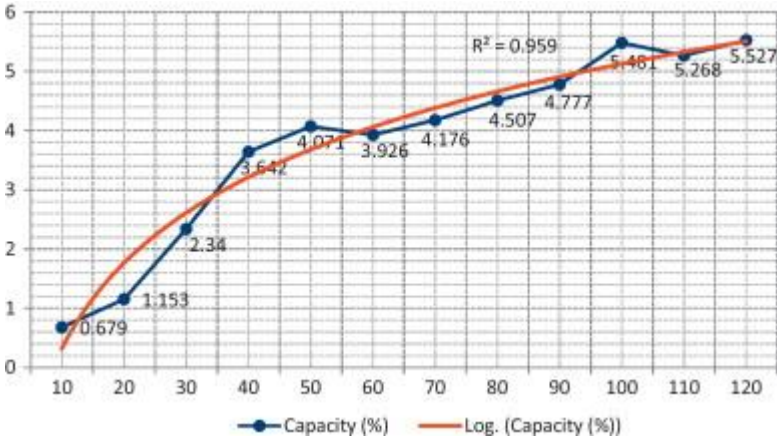
"By employing stego keys and Combinatorics-based coding security of the proposed method has been supported. One advantage of the proposed method is not being language specific and protecting the originality of cover text. Cover texts have been generated naturally, so they do not raise suspicion and the method is resilient against the possible attacks."[26]

In short the scheme works by implementing two features on top of LZW compression, firstly a dictionary stores common compression patterns to help reduce repetition and a stego key is shared with a random aspect to verify the identity of each party. The source classifies capacity as $c = \frac{\text{bitsOfSecretMessage}}{\text{bitsOfStegoText}}$ [26] using this

classification the source can infer capacity potential where (S) is the secret message, (n) is the number of characters, and (p) is the number of double or more repetitions. Generally, this will mean as S and n increase so too does the possibility of P becoming much greater

“As the length (n) of S increases, it can be seen that dual pattern repetition number (p) increases, too. This has a positive contribution to LZW coding which performs compression process by basing on symbol repetition.”[26]

The developed algorithm does not benefit from repetition although this could be a consideration worth investigation, by assigning a repeated letter or double in the case of common English text like ‘oo’, ‘ou’, ‘ly’ the embedding can be simplified to a typical symbol for these combinations increasing capacity per line to 2-Char/line at max. The capacity vs length of this scheme reflects the equation behind the algorithm’s logic showing an improvement trend in duplication over greater lengths of text.



(Fig. 3.8)Graph of capacity trend capacity vs. length[26]

See Below for comparative table

<i>Algorithm</i>	<i>Type</i>	<i>Capacity</i>	<i>Comments</i>
<i>Developed Algorithm[41]</i>	Mixed character/string end of line substitution	1(x)-Char/Line	Potential for more characters per line as the symbol correspondence can be redefined
<i>An overview of text steganography [10](Char String mapping)</i>	Character Level replacement technique	2-Bits/Cover Char	Binary embedding technique allows for greater variety of characters when utilising a charset
<i>An overview of text steganography [10](End of Line spacing)</i>	End of Line substitution	2-Bits/Line	Basis of developed algorithm uses a binary scheme but offers no aspect of security
<i>A Novel Text Steganography Technique Based on Html Documents by M.G [16]</i>	Mixed tag mapping technique	1-Char/HTML tag	Can encode a longer string of binary but at risk of document lacking attribute tags that detracts capacity.
<i>A compression-based text steganography method by E.S and H.I[26]</i>	Compression based LSZ technique	$c = \frac{bitsOfSecretMessage}{bitsOfStegoText}$	Offers key sharing for security and logs duplication in a dictionary

(Fig3.9) Table of listed and compared techniques[10][16][26][41]

3.3 Discussion

In comparison to other approaches in steganography it is important to consider the use case of each, examples which involve a lot of structure around the English language are not best applied to HTML formats. It appears that many techniques use a 2-bit encoding scheme which is then applied to a charset, this is because of the range of values this provides. The developed algorithm does not use this technique, instead opting to split a range of allowed value that can be freely mixed, from a steganography perspective this offers two main benefits. Firstly, the ability to vary order, size, and type of value that can be encoded. Secondly is used to optimize and obfuscate the key factor in ordering characters considered a light security factor. Other algorithms can provide better capacity for this reason but do not allow mixing in this varied format. In order to produce an effective modern steganography technique an aspect of security is expected, the developed algorithm provides this via character mixing based upon assignment to an ‘*encsign*.’ The developed algorithm could provide unique benefits over other similar analysed algorithms, also showing

good efficiency with an understandable and concise code base, despite these benefits continued development of the algorithm is advisable as many features that are not in the final build could be implemented and tested to improve the software (see. 4.2 Future Work). The next section will transfer the information from comparative analysis and this discussion into more applicable information for this and potential later projects.

3.4 Findings

The results of these analysis types can be processed into five core findings founded from the technical and comparative analysis of the software that will have a significant impact in the continued development of the algorithm.

- *Security is a necessary aspect in all modern competing steganography algorithms.*
Given that all modern steganography techniques are in competition with cryptography introducing security to steganography is a logical step, this can be seen from numerous sources examined such as *Esra Satir* and *Hakan Isik* LZW based technique which introduced keys as verification[26] or *Neil F. Johnson, Zoran Duric,* and *Sushil Jajodia's* continued discussion and examination of watermarking robustness and resilience[25]. A security mechanism can help algorithms ensure a first line of defence against attacks; this becomes more important in commercial aspects like watermarking as the potential cost of failure increases for the party using the algorithm.

- *2-Bit encoding is a common format but prevents individual assignment when using a charset.*

A core aspect of the developed algorithm is the ability to change assignment of markers and the groups of text they control, this unique aspect is rarely adopted as a greater variety and compatibility of characters can be obtained through using a 2-Bit binary scheme which then maps to a charset present in many common schemes documented by *An overview of text steganography*[10]. This may raise a security concern as if the stego chars are identified it will be easy to discern which charset is being used or launch a brute force attack to check each charset. This approach to embedding could be better secured by using mixing to remove similarities to any particular charset, the developed algorithm attempts to do this successfully by grouping and ordering of three separate alphabet groups and symbol/number groups. Once identified this will be harder to identify and map a pattern, especially if the plaintext is of small sample or includes an intentional misspelling.

- *Commonly occurring strings can be grouped to increase capacity.*

As documented by LZW[26] schemes common letters are grouped and added to a dictionary, this helps to reduce processing and increase capacity as this dictionary can be easily referenced when recurring characters are found. So long as the

language being used has no rules restricting all recurring characters this can be applied, notably if applied to the developed algorithm this concept can be taken to another level by including connectives or common phrases of the English language. However, this may compromise security as it can reduce complexity by overriding the mixing security aspect.

•*Successful schemes rely on off common frequent features of the entire document.*

To fully take advantage of a file a constant capacity should be established, this means that the capacity between documents of the same length should be the same, to restrict algorithms capacity by particular rare features will diminish it's use for shorter files but has the potential of greater secrecy so a balance must be met. For example *Mohit Garg's* algorithm[16] uses attribute ordering to create a binary correspondence for embedding, while this is a discrete solution it will rely on each line containing multiple attributes to maximise capacity, this will limit the HTML files it will apply to and is more suited to realistic online examples.

•*Secrecy VS. Capacity arguments.*

The previous findings bring up debates about secrecy and capacity, it is common occurrence that either one of these attributes is inversely proportional; the more secrecy the less capacity, the more capacity the less secrecy. This summary comes from file editing, the more changes made to a file the easier it will be to detect hence why the more data hidden the more must be manipulated to best fit this. The exception to this rule is steganography techniques that utilize metadata to hide a message[50] even in these cases automatic file checking can be used to reveal discrepancies making it a careful consideration if '*overly*' secretive techniques may be too conspicuous perhaps against the ethos of '*hiding in plain sight*'.

3.5 Summary

Overall the developed algorithm performs all functions within acceptable time and displays evidence of good efficiency when tasked with handling larger datasets especially where web HTML is concerned. The developed interface is designed to be simple and act as a one-to-one translation of the available functions of the program, the interface also enforces rules effectively ensuring compatibility and legality. In comparison to other algorithms the developed algorithm does not show as broad of a range of available characters instead opting for a smaller grouping and splitting to provide security. Other more advanced methods have provided greater insight into improvements that can be made for compatibility and capacity such as decreasing the grouping restriction and combining common characters to increase capacity over longer strings.

End of Chapter 3

Chapter 4

Conclusion and Future Work

This section will conclude all prior information and advise on potential adaptations that could be made to the algorithm for further improvements during continued development.

4.1 Conclusion

In conclusion steganography is a broad field, concentration on HTML based steganography still contains a wealth of information and current research on techniques which can be broadly transferred and techniques more focused on the language precisely. Based upon the initial research question *‘Can an algorithm that encodes a HTML file be created that fulfils three requirements has a low level of detection, maintains the structure and integrity of the cover text, and successfully encodes a variety of complex messages’*

The developed algorithm for embedding messages using end-of-line embedding achieves each of these requirements to varying degrees, the algorithm is concise, secretive and has no visual appearance, the requirement of embedding a variety of messages could be better covered using findings from research. The research conducted illustrates the importance of security in steganography and the variety of methods available for adaptation, the methodology used within this work helped place the developed algorithm in context of these findings but led to a quick discovery of the limiting factors of each requirement being tied together where more of one aspect usually meant a decrease in the other factors. This raises the question of context and prioritisation when developing a steganography algorithm. Despite this the research and development of a new unique algorithm has contributed by providing a Secretive, effective, and compatible HTML steganography algorithm presented through a simple user interface, produced through thorough review of a great variety of research in general and specific sub-fields.

4.2 Future Work

Although successful by the requirements the developed algorithm could be further improved with information learned from development, research, and analysis.

The top priority for improvement before adding more features is to optimise the checks performed when embedding a character, although the program executes in acceptable and responsive time optimising will simplify checks and make implementation of later features easier and increase compatibility for a greater range of data. Optimisation would verify the number of total checks needed and seek to reduce evaluations in relation to selection to reduce the efficiency k value.

The algorithm also relies upon character grouping, in the development future a mixing system that maintains security yet applies to a charset to allow most characters would benefit the systems compatibility across all languages and allow for deeper meta embedding should the user choose to embed within the plaintext initially.

To aid in capacity other algorithms have chosen to group common characters to increase capacity, this could be applied to the algorithm to reduce the space consumed at the end of each line making the embedding have a smaller surface area.

These various improvements in continued development will be investigate and further documented before being published to the repository.[41]

End of Chapter 4

Bibliography

- [1] *Fridrich, J., Goljan, M., Soukal, D., & Department of Computer Science, SUNY Binghamton. (2004, June). Searching for the Stego-Key. binghamton.edu. Retrieved 15 August 2022, from*
(http://www.ws.binghamton.edu/fridrich/Research/Keysearch_SPIE.pdf)

- [2] *Garg, M. G. (2011). A Novel Text Steganography Technique Based on Html Documents. International Journal of Advanced Science and Technology, 35(1), 129–138.*
(<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.445.2007&rep=rep1&type=pdf>)

- [3] *Odeh A., Elleithy K., Faezipour M., Abdelfattah E. (2015) Novel Steganography over HTML Code. In: Sobh T., Elleithy K. (eds) Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering. Lecture Notes in Electrical Engineering, vol 313. Springer, Cham. https://doi.org/10.1007/978-3-319-06773-5_81*

- [4] *Shahreza M.S. (2007) A New Method for Steganography in HTML Files. In: Elleithy K., Sobh T., Mahmood A., Iskander M., Karim M. (eds) Advances in Computer, Information, and Systems Sciences, and Engineering. Springer, Dordrecht. https://doi.org/10.1007/1-4020-5261-8_39*

- [5] *Wikipedia contributors. (2001, November 12). Cryptography. Wikipedia. Retrieved 22 September 2022, from*
(<https://en.wikipedia.org/wiki/Cryptography>)

- [6] *Newman, L. H. (2022, June 8). How a Saxophonist Tricked the KGB by Encrypting Secrets in Music. Wired. Retrieved 22 September 2022, from*
(<https://web.archive.org/web/20220608222058/https://www.wired.com/story/merryl-goldberg-music-encryption-ussr-phantom-orchestra>)

- [7] *Ciphers, Codes, & Steganography*. (2015, January 23). Folger Shakespeare Library. Retrieved 8 August 2022, from <https://www.folger.edu/ciphers-codes-steganography>
- [8] Johann, F. (2004). *Cryptographia(1684) [Hardback]*. In *Cryptographia : oder Geheime schrift- münd- und würckliche Correspondentz (1st ed.)*. Rebenlein Georg. <https://dbc.wroc.pl/dlibra/publication/144214/edition/75599> (Original work published 1684) oai:dbc.wroc.pl:75599
- [9] Anderson, R. (1997, January 1). *Information Hiding: First International Workshop, Cambridge, U.K., May 30 - June 1, 1996. Proceedings*. Springer.
- [10] R. B. Krishnan, P. K. Thandra and M. S. Baba, "An overview of text steganography," 2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN), 2017, pp. 1-6, doi: 10.1109/ICSCN.2017.8085643.
- [11] *Digital Watermarking and Steganography, 2nd Ed. (The Morgan Kaufmann Series in Multimedia Information and Systems) (2nd ed.)*. (2007, November 27). [Paperback]. Morgan Kaufmann.
- [12] M. H. Shirali-Shahreza and M. Shirali-Shahreza, "Text Steganography in chat," 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet, 2007, pp. 1-5, doi: 10.1109/CANET.2007.4401716.
- [13] M. Shirali-Shahreza, "Text Steganography by Changing Words Spelling," 2008 10th International Conference on Advanced Communication Technology, 2008, pp. 1912-1913, doi: 10.1109/ICACT.2008.4494159.
- [14] Dulera, S., Jinwala, D., & Dasgupta, A. (2011, November 30). *Experimenting with the Novel Approaches in Text Steganography*. *International Journal of Network Security & Its Applications*, 3(6), 213–225. <https://doi.org/10.5121/ijnsa.2011.3616>
- [15] S. Koley and K. K. Mandal, "A novel approach of secret message passing through text steganography," 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), 2016, pp. 1164-1169, doi: 10.1109/SCOPES.2016.7955624.

- [16] Garg, M. (2011, October). *A Novel Text Steganography Technique Based on Html Documents*. *International Journal of Advanced Science and Technology*, 35, 129–138.
(<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.445.2007&rep=rep1&type=pdf>)
- [17] Odeh, A., Elleithy, K., Faezipour, M., Abdelfattah, E. (2015). *Novel Steganography over HTML Code*. In: Sobh, T., Elleithy, K. (eds) *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering. Lecture Notes in Electrical Engineering*, vol 313. Springer, Cham.
https://doi.org/10.1007/978-3-319-06773-5_81
- [18] Shahreza, M. S. (n.d.). *A New Method for Steganography in HTML Files*. *Advances in Computer, Information, and Systems Sciences, and Engineering*, 247–252.
https://doi.org/10.1007/1-4020-5261-8_39
- [19] K. F. Rafat, "Enhanced text steganography in SMS," 2009 2nd International Conference on Computer, Control and Communication, 2009, pp. 1-6, doi: 10.1109/IC4.2009.4909228.
- [20] D. Artz, "Digital steganography: hiding data within data," in *IEEE Internet Computing*, vol. 5, no. 3, pp. 75-80, May-June 2001, doi: 10.1109/4236.935180.
- [21] Por, L. Y., & Delina, B. (n.d.). *Information Hiding: A New Approach in Text Steganography*. In *ACACOS*.
(https://www.researchgate.net/publication/228623052_Information_hiding_A_new_approach_in_text_steganography)
- [22] Majeed, M. A., Sulaiman, R., Shukur, Z., & Hasan, M. K. (2021, November 8). *A Review on Text Steganography Techniques*. *Mathematics*, 9(21), 2829.
<https://doi.org/10.3390/math9212829>
- [23] Doshi, R., Jain, P., & Gupta, L. (Eds.). (2012, November). *Steganography and Its Applications in Security* (6th ed., Vol. 2). *International Journal of Modern Engineering Research (IJMER)*.
(<http://iotresaneh.ir/wpcontent/uploads/2020/05/Steganography.pdf>)

- [24] Shih, F. Y. (2017, April 26). *Digital Watermarking and Steganography: Fundamentals and Techniques, Second Edition (2nd ed.)*. CRC Press.
- [25] Johnson, N. F., Duric, Z., & Jajodia, S. (2000, December 31). *Information Hiding: Steganography and Watermarking - Attacks and Countermeasures (2001st ed.)*. Springer.
- [26] Esra Satir, Hakan Isik, *A compression-based text steganography method*, *Journal of Systems and Software*, Volume 85, Issue 10, 2012, Pages 2385-2394, ISSN 0164-1212, <https://doi.org/10.1016/j.jss.2012.05.027>.
(<https://www.sciencedirect.com/science/article/pii/S0164121212001379>)
- [27] Thabit, R., Udzir, N. I., Yasin, S. M., Asmawi, A., Roslan, N. A., & Din, R. (2021, July 26). *A Comparative Analysis of Arabic Text Steganography*. *Applied Sciences*, 11(15), 6851. <https://doi.org/10.3390/app11156851>
- [28] Fridrich, J. (2012, November 8). *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press.
- [29] Elżbieta Zielińska, Wojciech Mazurczyk, and Krzysztof Szczypiorski. 2014. *Trends in steganography*. *Commun. ACM* 57, 3 (March 2014), 86–95.
<https://doi.org/10.1145/2566590.2566610>
- [30] Singh, N. (2017, January). *Survey Paper on Steganography*. *International Refereed Journal of Engineering and Science*, 6(1), 68–71.
(<http://www.irjes.com/Papers/vol6-issue1/K616871.pdf>)
- [31] Anderson, R. (1997b, January 1). *Information Hiding: First International Workshop, Cambridge, U.K., May 30 - June 1, 1996. Proceedings*. Springer.
- [32] Lip Yee Por, KokSheik Wong, Kok Onn Chee, *UniSpaCh: A text-based data hiding method using Unicode space characters*, *Journal of Systems and Software*, Volume 85, Issue 5, 2012, Pages 1075-1082, ISSN 0164 1212,
<https://doi.org/10.1016/j.jss.2011.12.023>.
(<https://www.sciencedirect.com/science/article/pii/S0164121211003177>)

- [33] *Gallagher, N. (2012, May). Principles of writing consistent, idiomatic HTML. GitHub. Retrieved 22 July 2022, from*
(<https://github.com/necolas/idiomatic-html>)

- [34] *ASCII Table - ASCII Character Codes, HTML, Octal, Hex, Decimal. (n.d.). Ascitable. Retrieved 18 August 2022, from*
(<https://www.asciitable.com>)

- [35] *GitHub: Where the world builds software. (n.d.). GitHub.*
(<https://github.com>)

- [36] *GeeksforGeeks. (2021, November 8). LZW (Lempel–Ziv–Welch) Compression technique. Retrieved 24 August 2022, from*
(<https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique>)

- [37] *Google. (n.d.). Google steganography trends. Google Trends. Retrieved 4 September 2022, from*
(<https://trends.google.com/trends/explore?date=all&q=%2Fm%2F073mt,%2Fm%2F01ld0>)

- [38] *Python. (n.d.). Python Release Python 3.8.0. Python.org. Retrieved 8 July 2022, from*
(<https://www.python.org/downloads/release/python-380>)

- [39] *JetBrains. (2021, June 2). PyCharm: the Python IDE for Professional Developers. Retrieved 7 July 2022, from*
(<https://www.jetbrains.com/pycharm>)

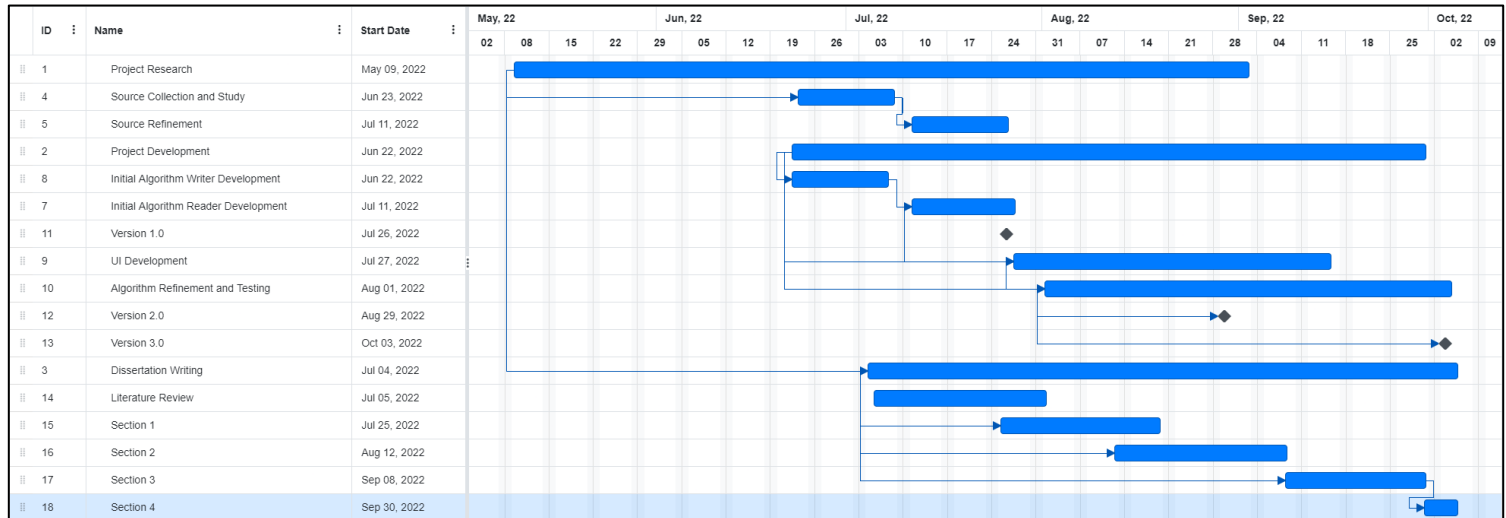
- [40] *RiverBank Computing. (n.d.). PyQt5. PyPI. Retrieved 12 August 2022, from*
(<https://pypi.org/project/PyQt5>)

- [41] *Norman, J. (2022, September 30). Personal Repository. GitHub.*
(<https://github.com/Ju5t-j1m/Y4project>)

- [42] *GoOnlineTools. (n.d.). Online HTML Viewer | GoOnlineTools. GO Online Tools. Retrieved 1 September 2022, from (<https://goonlinetools.com/html-viewer>)*
- [43] *Adobe. (2022, March 18). Waterfall Methodology. Adobe Experience Cloud Blog. (<https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%E2%80%94also%20known,before%20the%20next%20phase%20begins>)*
- [44] *MDN. (2022, September 15). HTML basics - Learn web development | MDN. Mdn Web Docs. (https://developer.mozilla.org/enUS/docs/Learn/Getting_started_with_the_web/HTML_basics)*
- [45] *Rapid. (n.d.). Unicode characters table. RapidTables. (<https://www.rapidtables.com/code/text/unicode-characters.html>)*
- [46] *gaogaotiantian. (n.d.). VizTracer is a low-overhead logging/debugging/profiling tool that can trace and visualize your python code execution. GitHub. Retrieved 9 September 2022, from (<https://github.com/gaogaotiantian/viztracer>)*
- [47] *Fuchs, C. (2022, April 16). UX User Experience Management - Application of a Usability Engineering Lifecycle: Concepts and methods for the engineering production of user-friendliness or usability. Independently published.*
- [48] *ICG. (n.d.). ICG Agency. Retrieved 23 September 2022, from (<https://www.icg.agency/blog/whats-the-maximum-file-size-google-can-index>)*
- [49] *UXPin. (n.d.). The Basic Principles of User Interface Design. Studio by UXPin. Retrieved 31 August 2022, from (<https://www.uxpin.com/studio/blog/ui-design-principles>)*
- [50] *Castiglione, A., De Santis, A., & Soriente, C. (2007, May). Taking advantages of a disadvantage: Digital forensics and steganography using document metadata. Journal of Systems and Software, 80(5), 750–764. doi.org/10.1016/j.jss.2006.07.006 (<https://www.sciencedirect.com/science/article/pii/S0164121206001981>)*

Appendix A

Time Cost Gantt Chart



Listing A.1: Gantt Chart of concurrent activities across development

ID	Name	Start Date	End Date	Duration	Dependency
1	Project Research	May 09, 2022	Sep 02, 2022	85 days	
4	Source Collection and Study	Jun 23, 2022	Jul 08, 2022	12 days	1SS+33 days
5	Source Refinement	Jul 11, 2022	Jul 26, 2022	12 days	4FS
2	Project Development	Jun 22, 2022	Sep 30, 2022	73 days	
8	Initial Algorithm Writer Development	Jun 22, 2022	Jul 07, 2022	12 days	2SS
7	Initial Algorithm Reader Development	Jul 11, 2022	Jul 27, 2022	13 days	8FS+1 day
11	Version 1.0	Jul 26, 2022	Jul 26, 2022	0 days	
9	UI Development	Jul 27, 2022	Sep 15, 2022	37 days	8SS+25 days,7SS+12 d...
10	Algorithm Refinement and Testing	Aug 01, 2022	Oct 04, 2022	47 days	9SS+3 days,2SS+28 days
12	Version 2.0	Aug 29, 2022	Aug 29, 2022	0 days	10SS+21 days
13	Version 3.0	Oct 03, 2022	Oct 03, 2022	0 days	10SS+46 days
3	Dissertation Writing	Jul 04, 2022	Oct 05, 2022	68 days	1SS+40 days
14	Literature Review	Jul 05, 2022	Aug 01, 2022	20 days	
15	Section 1	Jul 25, 2022	Aug 19, 2022	20 days	3SS+15 days
16	Section 2	Aug 12, 2022	Sep 08, 2022	20 days	3SS+29 days
17	Section 3	Sep 08, 2022	Sep 30, 2022	17 days	3SS+48 days
18	Section 4	Sep 30, 2022	Oct 05, 2022	4 days	17FS-1 days

Listing A.2: Gantt Chart Time slices and durations

Appendix B

Writer and Reader Specific Algorithms

Writer.py (line 67-100)

```
for f, b in itertools.zip_longest(split, plain_list, fillvalue='!'):

    # First Third of Alphabet from e - r (LFA order)#
    if f.endswith(('>', '"', "\t")) and not b.isupper() and 1 <= rebuilt_kb.index(b) <= 9:
        temp = rebuilt_kb.index(b) # power value for space
        f = (f + (steg_char_value * temp)) # combining to the end of the string
        result.append(f)

    # Second Third of Alphabet from r - y #
    elif f.endswith(('>', '"', "\t")) and not b.isupper() and 10 <= rebuilt_kb.index(b) <= 18:
        temp = (rebuilt_kb.index(b) - 10) # offset by 10 to start a new batch of up to 8 spaces
        f = (f + "\t" + (steg_char_value * temp)) # encoded with a tab marker
        result.append(f)

    # Third Third of Alphabet from p - z #
    elif f.endswith(('>', '"', "\t")) and not b.isupper() and 19 <= rebuilt_kb.index(b) <= 26:
        temp = (rebuilt_kb.index(b) - 19)
        f = (f + "\t" + (steg_char_value * temp)) # encoded with a tab space marker
        result.append(f)

    # Digits and Symbols from 0..9 - '!.'. first third type removes kb- #
    elif f.endswith(('>', '"', "\t")) and not b.isupper() and 28 <= rebuilt_kb.index(b) <= 54:
        temp = (rebuilt_kb.index(b) - (len(kb_logical_blank) + len(kb1_alphabet)))
        f = (f + (steg_char_value * 9) + (steg_char_value * temp))
        result.append(f) # will evaluate as any space > 9

    # Capital Letters uses a fake comment as a marker #
    elif f.endswith(('>', '"', "\t")) and b.isupper():
        temp = (rebuilt_kb.index(b)) - len(rebuilt_kb) + len(kb4_uppercase) + 1
        f = (f + " <!--Auto Stub-->" + (steg_char_value * temp))
        result.append(f)

    else: # appends any line that can't or does not need processing
        result.append(f)
```

Listing B.1: Writer algorithm (shows filtering for knowledge base characters).

Reader.py (line 33-67)

```
For s in range(len(splitSteg)):

    # search for encsign in each string
    find2 = splitSteg[s].find(encsign2)
    find3 = splitSteg[s].find(encsign3)
    steg_char_num = (splitSteg[s].rfind('>') + 1) # place reader 1 position ahead

    # Third Third of alphabet #
    if find2 >= 0:
        steg_char_num = len(splitSteg[s]) - len(encsign2)
        len(splitSteg[s].rstrip(Writer.steg_char_value)) # len
        plaintext.append(rebuilt_kb[steg_char_num + third_alphabet_offset]) # offset for first space char

    # First Third and Digit/Symbol find the sign at the length of plain line + 1 #
    elif splitSteg[s].endswith(encsign, steg_char_num, steg_char_num + 1) and find2 == -1 and find3 == -1:
        steg_char_num = len(splitSteg[s]) - len(splitSteg[s].rstrip(Writer.steg_char_value))

    # Digit and Symbol check #
    if steg_char_num >= (digit_symbol_marker + 1): # same style where first third drops at a max of 9
        plaintext.append(digit_symbol_kb[steg_char_num - digit_symbol_marker]) # removal of marker

    else:
        plaintext.append(rebuilt_kb[steg_char_num]) # otherwise this is an alphabet char

    # Second Third of Alphabet similar check but for encsign1 instead evaluated later #
    elif splitSteg[s].endswith(encsign1, steg_char_num, steg_char_num + 1) and find2 == -1 and find3 == -1:
        steg_char_num = len(splitSteg[s]) - len(encsign1) - len(splitSteg[s].rstrip(Writer.steg_char_value))
        plaintext.append(rebuilt_kb[steg_char_num + second_alphabet_offset]) # offset of next alphabet chars

    # Capital Letters #
    elif find3 >= 0: # removes dummy text length
        steg_char_num = len(splitSteg[s]) - len(encsign3) - len(splitSteg[s].rstrip(Writer.steg_char_value))
        plaintext.append(Writer.kb4_uppercase[steg_char_num + capital_letter_offset])

    else:
        result = ".join(plaintext) # final building of decoded text
        return str(result)
```

Listing B.2: Reader algorithm (corresponds to *B.1* and shows reading order).

Full Code available at: <https://github.com/Ju5t-j1m/Y4project>