

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Triinu Erik 164843IAPB

STEGOTE – STEGANOGRAPHY TOOL FOR HIDING INFORMATION IN JPEG AND PNG IMAGES

Bachelor's thesis

Supervisor: Sten Mäses

MSc

Co-supervisor: Rémi Cogranne

PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Triinu Erik 164843IAPB

STEGOTE – STEGANOGRAAFIA TÖÖRIIST JPEG JA PNG PILTIDESSE INFO PEITMISEKS

Bakalaureusetöö

Juhendaja: Sten Mäses

MSc

Kaasjuhendaja: Rémi Cogranne

PhD

Tallinn 2019

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Triinu Erik

21.08.2019

Abstract

The goal of this thesis is to create a customizable steganography tool called Stegote that allows users to hide data into digital images. The users need to be able to choose the way their data is hidden. Stegote has to hide data into JPEG and PNG images in an undetectable manner, using two different LSB embedding methods and three different path generation methods. The tool is open-source.

This thesis describes the realization process of Stegote and analyses five other popular steganography tools and compares them to Stegote, assuring that Stegote offers the highest degree of customizability. Additionally, Stegote is steganalysed in order to verify the steganography's undetectability and that steganographically modified images are not differentiable from regular images. Stegote's UI/UX is tested with a usability test.

This thesis is written in English and is 31 pages long, including 7 chapters, 24 figures and 2 tables.

Annotatsioon

Stegote – steganograafia tööriist JPEG ja PNG piltidesse info peitmiseks

Käesoleva töö põhieesmärgiks on luua steganograafia tööriist nimega Stegote, mis võimaldab kasutajatel peita infot digitaalsetesse piltidesse. Steganograafia tähendab informatsiooni peitmist mingi teise objekti sisse, millega võimaldatakse hoida saladuses nii sõnumi sisu kui ka tõsiasja, et sõnumit üldsegi saadeti.

Loodav tööriist peab võimaldama kasutajal peitmise viisi valida ning peitma infot nii, et seda poleks võimalik tuvastada paremini kui juhusliku oletuse tõenäosusega. Stegote peidab infot nii JPEG kui PNG piltidesse, kasutades selleks meetodit, mis peidab info vähima kaaluga bittidesse. Stegote kasutab kahte erinevat vähima kaaluga biti sisestamise võtet ning kolme erinevat teekonna genereerimise algoritmi. Stegote on avatud lähtekoodiga.

Bakalaureusetöö raames kirjeldatakse Stegote realisatsiooni protsessi ning analüüsitakse viit teist populaarset steganograafia tööriista ning võrreldakse neid Stegotega. Selle käigus veendutakse, et tööpoolest pakub Stegote kõige rohkem valikuvõimalusi info peitmise viisi osas. Samuti steganalüüsitakse Stegoted eesmärgiga veenduda, et peidetud infoga pilte pole võimalik eristada tavalistest piltidest. Stegote kasutajaliidest ja kasutajakogemust testitakse kasutatavuse testiga.

Lisades antakse põhjalik teoreetiline ülevaade bakalaureusetöö raames kasutatud tehnikatest ja kontseptsioonidest: pakkimisest ja JPEG pakkimise standardist ning selle implementeerimise etappidest, steganograafiast ja vähima kaaluga bittide sisestamisest ning steganalüüsimisest.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 31 leheküljel, 7 peatükki, 24 joonist, 2 tabelit.

List of abbreviations and terms

AC	Coefficient with non-zero frequencies
AU	Audio file format
BMP	Bitmap image format
DC	Coefficient with zero frequency
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
FPR	False Positive Rate
G-LSB	Generalized-LSB
GUI	Graphical User Interface
HVS	Human Visual System
IDCT	Inverse Discrete Cosine Transform
JAR	Java Archive file
JPEG / JPG	Joint Photographic Experts Group
JPEG image	Image that is JPEG compressed: steganography with JPEG images uses the quantized DCT coefficients of the image
LED	Light Emitting Diodes
LSB	Least Significant Bit
Plain image	Image that is not compressed: steganography with a plain image uses the RGB plane of the image.
PNG	Portable Network Graphics
PSNR	Peak Signal to Noise Ratio
RGB	Red, Green, Blue colour model
RLE	Run Length Encoding
ROC	Receiver Operating Characteristic
Steganalysis	The activity of trying to detect steganography [1].
TalTech	Tallinn University of Technology
TPR	True Positive Rate

UI	User Interface
UX	User Experience
WAV	Waveform Audio file format
YCbCr	Luminance, Red and Blue Chrominance colour model

Table of contents

1 Introduction	13
1.1 Problem statement	14
1.2 Contribution.....	14
1.3 Structure of the thesis	15
2 Related work.....	16
2.1 Trends in LSB embedding techniques.....	16
2.2 Similar solutions	18
3 Requirements	21
4 Realization.....	23
4.1 Technical decisions.....	23
4.2 Realization of JPEG compression	24
4.3 Realization of path generation.....	26
4.3.1 Generating a simple path for a plain image.....	27
4.3.2 Generating a simple path for a JPEG image.....	27
4.3.3 Generating a path with a shared key for a plain image.	28
4.3.4 Generating a path with a shared key for a JPEG image	28
4.3.5 Generating a path encrypted with a secret key for a pain image.....	29
4.3.6 Generating a path encrypted with a secret key for a JPEG image.....	30
4.4 Realization of LSB embedding.....	30
4.5 User interface.....	32
5 Validation	36
5.1 Comparison with similar solutions	36
5.2 Steganalysis on Stegote	37
5.3 Usability testing.....	40
6 Limitations and future work	42
7 Conclusion.....	43
References	44
Appendix 1 – Compression	47
Image compression.....	47

Why is image compression needed?.....	47
How is image compression possible?.....	48
Psychovisual interpretation	49
Lossy and lossless compression	50
JPEG compression.....	51
Colour transformation	52
Division into blocks and subsampling.....	54
DCT transform.....	55
Quantization	56
Encoding and lossless compression.....	57
Appendix 2 – Steganography	59
Steganographic system	59
Steganography paradigms.....	61
Steganography by cover modification.....	62
LSB embedding	64
LSB replacement	64
LSB matching.....	65
Decoding LSB embedded messages.....	65
Appendix 3 – Steganalysis	67
ROC curve	68
Appendix 4 – Usability testing tasks	70

List of figures

Figure 1. Hiding process of a secret message into a cover image. Blue parts represent the encoding, red parts represent JPEG compression.....	22
Figure 2. Example of quantized DCT coefficients.....	25
Figure 3. View after entering the --help command.....	32
Figure 4. Example of using the tool to encode a message.....	33
Figure 5. Example of a secret image with data embedded into it.....	33
Figure 6. Example of using the tool to decode a message.....	34
Figure 7. Example of a decoded secret message.....	34
Figure 8. Example of generating a key.....	35
Figure 9. ROC curve of StegExpose tested against LSB-Steganography, OpenPuff, OpenStego and SilentEye.....	38
Figure 10. ROC curve of StegExpose tested against Stegote.....	40
Figure 11. Example: a portrait [15] where some pixels have been changed to carry unlikely values, i.e. dark pixels in the middle of a face and vice versa.....	49
Figure 12. Although they seem almost identical, the image on the right is ~80% smaller than the image on the left.....	51
Figure 13. When closely looked, the compressed image (right) has highly visible distortions compared to the original image (left).....	51
Figure 14. An image divided to its red, green and blue components [21].....	52
Figure 15. Visual representation [22] of the YCrCb model.....	53
Figure 16. RGB to YCrCb transformation visualized [21], presuming no subsampling has been done.....	54
Figure 17. The spatial frequency representation of DCT [24].....	56
Figure 18. The visual presentation [24] of the zig-zag algorithm on an 8×8 block.....	57
Figure 19. Visualisation of the elements of a steganographic system.....	60
Figure 20. Visualization of steganography by cover modification.....	63
Figure 21. Pseudo-code of LSB replacement.....	64
Figure 22. Pseudo-code of LSB matching.....	65
Figure 23. Pseudo-code of decoding LSB embedded message.....	66

Figure 24. Examples [20] of ROC curves. 69

List of tables

Table 1. Comparison of five major tools and the author's tool, Stegote.	36
Table 2. True and false positives, TPR and FPR for selected thresholds for the StegExpose tool used against the author's tool, Stegote.....	39

1 Introduction

There are occurrences where it might be necessary to communicate some information in a secret way, so that no one else but the communicating partners is able to understand the meaning. This could be sensitive or secret information, which for some reason or another has to stay concealed. At the same time, this communication has to often take place over a public medium, where the message could be read by someone it was not meant for. This means that the obfuscated data is assumed to be accessible and readable by third parties, but the meaning it carries should not, at the same time, be understood. To achieve that, there are generally two ways:

1. Cryptography
2. Steganography

Cryptography is efficient and it is great to preserve the secrecy of the message [2]. For example, communicating partners can encrypt and decrypt the message using a shared key that only they know. When the encryption algorithm used is strong enough, even if the encrypted message is read by third parties, it is not considered a threat to the secrecy of the message.

On the other hand, cryptography has a downside of being very easily detectable. That means, even though the meaning of the message is not understood, it is clear that a secret communication is happening and it is known who writes whom. This can be called side information. In some cases, even this side information cannot be known; the side information is already revealing too much [3].

When there is a need to conceal the fact that there even is any secret communication happening, it is useful to use steganography. Steganography is the practice of concealing information in some other object. When the communication is happening over the Internet, digital media is an ideal medium. Images, videos, audio files etc. are frequently exchanged over the Internet, which means communicating by using them will not raise suspicion. Especially digital images are the perfect medium because there are

massive amounts of images on the Internet and they can be very easily sent and exchanged. Furthermore, images will mostly not be processed by the service provider the message was sent with (unlike uploading, for example, video files) and it's difficult to detect any hidden data in them without specialized tools.

1.1 Problem statement

Many popular freely available steganography tools can be considered cracked [4], which means that the presence of secret information hidden with those tools can be fairly reliably detected using steganalysis. In many cases, these tools are used as generators to test steganalytical methods against them. Also, they offer low customizability in their embedding strategies, meaning that they always hide the message using the same method. Thus, once a tool like this is cracked, it cannot be safely used again.

The aim of this thesis is to develop a highly customizable steganography tool that enables users to have a high degree of choice in the way their data is hidden. The tool should hide data into digital images, using JPEG and PNG file formats. The produced images should not be distinguishable from regular images.

In addition, the tool helps the co-supervisor in his research in the University of Technology of Troyes. He also intends to use the tool in two of his courses on cyber security.

The tool is open-source and freely available on Github¹.

1.2 Contribution

During this thesis, the author created a steganography tool called Stegote to hide data into digital images. The user can choose between two file formats, three path generation algorithms and two embedding strategies, altogether offering ten different ways to hide data into digital images. For this, the author implemented the lossy part of JPEG compression, developed six path generation algorithms dependent on the file format

¹ <https://github.com/triinerik/stegote>

(PNG and JPG) and implemented LSB embedding for six different use cases. When taking into account the slightly different algorithms for colour and greyscale images, the author developed 20 different ways to hide data into images. In addition, the author created a comparative analysis with other tools, verified the undetectability of steganographic images with a steganalysis tool and carried out a usability test on Stegote.

1.3 Structure of the thesis

The thesis is composed of seven chapters: introduction, related work, requirements, realization, validation, limitations and future work and conclusion. In related work, some trends in LSB embedding techniques are discussed and some similar solutions to Stegote are brought out. In requirements, the main needs and requirements for Stegote are described. In realization, the technical decisions, realization process and user interface of Stegote is written out in detail. In validation, the validation of results is performed by comparing Stegote with similar solutions and steganalysing it with an analysis tool while also describing the results of usability testing. Finally before concluding the thesis, the limitations and future work on Stegote is brought out. In the first three appendixes, theoretical background on compression, steganography and steganalysis can be read, while the fourth contains the usability test cases.

2 Related work

Steganography, despite not being very novel, remains to be an important field of research. By hiding data in a cover object it is possible to maintain the confidentiality of valuable information and protect it from sabotage, theft or unauthorised viewing [5]. It is also important in countries where communication is monitored and encrypted messages are restricted [5]. Surprisingly, even though hiding data in an undetectable manner is typically the main goal of steganography, the opposite goal is approached when using steganography in watermarking. Watermarking is used against copyright infringements by imperceptibly and robustly embedding information in the digital image such that it cannot be removed [6].

There are many different strategies and techniques used to hide data into media. This thesis uses LSB embedding, namely LSB replacement and LSB matching. LSB embedding, LSB replacement and LSB matching are discussed in further detail in Appendix 2. But these are only few of the algorithms to hide data. In this chapter, some alternative LSB embedding strategies are discussed. In additions, five popular and easily available steganography tools are analysed.

2.1 Trends in LSB embedding techniques

As mentioned before, this thesis uses LSB replacement and LSB matching strategies in embedding bits into images. These are only two of the many LSB embedding strategies. As all LSB embedding algorithms can be read with the same decoder (in further detail in Appendix 2), it would not be difficult to implement other, alternative LSB embedding strategies. In this chapter, four of them are described.

Even though LSB embedding is one of the first and more simple ways to hide data into images, new algorithms are being proposed and LSB embedding continues to be a popular trend in steganography. In the article "Performance Comparison of Steganography Techniques" [7], the authors claim **that LSB embedding continues to**

be highly undetectable: "... it is found that the LSB steganography and LSB using secret key perform the best on the basis of PSNR", PSNR (Peak Signal to Noise Ratio) being the most commonly used parameter to measure the quality of image after embedding [7]. When using LSB embedding with the DCT coefficients of a JPEG compressed image, the detection rate is even smaller [8]. In addition, the embedding capacity LSB techniques is high [7]. In this section, recent trends are described by discussing some alternative LSB embedding methods and strategies.

Generalized-LSB (G-LSB) embedding [9] is a strategy based on LSB embedding. This technique modifies the lowest levels — instead of bit planes — of the host signal to accommodate the payload information [9]. In the article "Lossless Generalized-LSB data embedding" [9] the authors propose the G-LSB method in the following way: "In the embedding phase, the lowest L levels of the signal samples are replaced (overwritten) by the watermark payload using a quantization step followed by an addition. During extraction, the watermark payload is extracted by obtaining the quantization error — or simply reading lowest L levels — of the watermarked signal. The classical LSB modification, which embeds a binary symbol (bit) by overwriting the least significant bit of a signal sample, is a special case where $L = 2$. G-LSB embedding enables embedding of non-integer number of bits in each signal sample and, thus, introduces new operating points along the rate (capacity)-distortion curve."

The **F5** algorithm was originally designed to overcome the histogram attack (detection method based on analysing the histogram [10]) while still offering a large embedding capacity [11]. F5 is composed of two steps: the embedding operation and matrix embedding. Firstly, the algorithm embeds the message bits in the LSBs of DCT coefficients [12]. In the article "Relating the embedding efficiency of LSB Steganography techniques in Spatial and Transform domains" [12], the embedding operation is described in the following way: "If the coefficient's LSB needs to be displaced, instead of flipping the LSB, the absolute value of the DCT coefficient is reduced by one. To avoid introducing absolutely detectable artefacts, the F5 skips completely the DC terms along with other coefficients equal to 0." Then, as the second step, matrix embedding is utilised. Matrix embedding improves the embedding efficiency of a message [13]. Matrix embedding encodes the cover image and the secret message with an error correction code and modifies the cover image according to the coding result [13].

The **adaptive LSB** embedding algorithm follows a directional embedding technique for achieving maximum image quality in the steganographic image [14]. This method performs a selection of suitable direction for secret byte embedding so as to minimize the bit changes in the cover image when a secret data is embedded [14]. This is where the name of the method comes from, as the algorithm adapts to the cover image's LSBs in order to make less changes to them. A direction bit is added at the 9-th bit which indicates that the preceding data is in stored in a reverse order [14]. A value 0 for the direction bit indicates a normal forward direction of storing data while a value 1 for the direction bit indicates that the data is stored in reverse direction [14].

A very interesting and novel approach to steganography is **LSB rotation** [15]. In the article "LSB Rotation and Inversion Scoring Approach to Image Steganography" [15], the authors describe the method in the following way: "Prior embedding, the bits of each byte of the secret message will be rotated eight times in a sequence along with the indicator bits that signifies current rotation position and inversion status. The byte rotation generates eight different combinations of the secret message byte as candidates of replacement to the targeted least significant bits of the cover image. After its 8th rotation, all bits of the secret message byte are inverted, and then rotated and scored again eight more times. The inversion will produce new byte value of the secret message and the 2nd eight rotations will generate eight more new combinations in an attempt to find other candidates that has even have lower difference score. Out of the sixteen candidates, the one that has its combination that produced the lowest difference score will have its rotated value, rotate position, and inversion status recalled and then embedded into the steganographic image in a fixed four bits per byte replacement approach. Because of the numerous candidates generated for embedding selection, the probability of finding and selecting the least distorting combination of the secret message byte is highly increased, and therefore effectively minimizing the distortion of the steganographic image."

2.2 Similar solutions

This chapter compares and analyses a few more popular and easily available steganography tools for digital images similar to Stegote and their strengths and

weaknesses. These tools were chosen from the most popular results¹ when searching for steganography tools online. Then, further choice was made by how well documented the tool was and if the link provided was working (often, the link was broken). In addition, the author tried to choose a variety of tools in order to provide an overview of the different types of tools available (open-source tools, tools with a GUI, command-line tools, tools hiding into PNG or JPEG, etc.)

OpenStego² is a free open-source steganography solution that allows the user to hide a text message into a cover image. It also supports watermarking in beta. OpenStego is written in Java³. It is possible to use the functionalities either through a command-line tool using the JAR file or through a graphical user interface (GUI) that can be launched by using the bundled batch file or shell script. Thus, OpenStego can be launched on any OS as JAR files can be run on any system where the Java virtual machine exists. OpenStego uses LSB embedding. It seems that the project's author has intentions to add more algorithms in the future, as the algorithm is parameterizable (although for now there is only one option). OpenStego also provides a functionality to encrypt the message before embedding it.

Hide'N'Send⁴ is a free steganography tool that allows the user to hide a file inside a JPEG cover image. Hide'N'Send is only available on Windows operating systems (XP, Vista and 7). It has a simple GUI, but to launch it the .NET⁵ Framework 2.0 is needed. It is possible to parametrize the embedding algorithm, choosing either F5 or LSB embedding. Hide'N'Send also encrypts the file before hiding it.

SteganoG⁶ is a free steganography tool that allows the user to hide any file into a BMP image. SteganoG runs on only Windows operation systems (7, 8 and 10). It is created

¹ <https://resources.infosecinstitute.com/steganography-and-tools-to-perform-steganography/> ;
<https://www.greycampus.com/blog/information-security/top-must-have-tools-to-perform-steganography>

² <https://www.openstego.com/>

³ <https://www.java.com/>

⁴ https://download.cnet.com/Hide-N-Send/3000-2092_4-75728348.html

⁵ <https://dotnet.microsoft.com/>

⁶ <https://www.softpedia.com/get/PORTABLE-SOFTWARE/Security/Encrypting/Windows-Portable-Applications-Portable-SteganoG.shtml>

with Visual Basic and needs Visual Basic 6 runtime¹ to run. It has a powerful GUI with many options, for example it is possible to instantly send the file as an email or change the language settings. It provides the possibility to encrypt the file before hiding it. It is not possible to parametrize the hiding algorithms, only the encryption. The embedding algorithm is not disclosed.

Steghide² is a steganography tool that allows to not only hide data in images, but also audio files. Steghide supports JPEG, BMP, WAV and AU files. It is open-source and available for both Unix and Windows systems. Steghide requires a few libraries to be installed in order to compile or hide in certain file formats. Steghide uses a graph-theoretic approach to steganography. It uses a graph-theoretic matching algorithm that finds pairs of positions such that exchanging their values has the effect of embedding the corresponding part of the secret data [16]. If the algorithm cannot find any more such pairs all exchanges are actually performed [16]. This is the only algorithm Steghide uses and it is not possible to choose any other.

Jsteg³ is a steganography tool for hiding data into JPEG images. It is an open-source project that is written in Go⁴. Jsteg hides the data into the LSBs of JPEG compressed images. It is not possible to parametrize the hiding algorithms. Jsteg is a simple command-line tool that does not have a GUI. A simple "jsteg" command is included, which provides a simple wrapper around the package. Jsteg is available for using on all Unix and Windows operating systems.

¹ <https://www.microsoft.com/en-us/download/details.aspx?id=24417>

² <http://steghide.sourceforge.net/>

³ <https://github.com/lukechampine/jsteg>

⁴ <https://golang.org/>

3 Requirements

The main aim of the thesis was to create a Python program which allows to hide data into a digital image in an undetectable manner. The tool has to offer high customizability and allow the user to choose the way the data is hidden. The developed tool, called Stegote, would also help the supervisor move his research from MATLAB¹ to Python and he intends to use the steganography application in two of his courses he teaches in University of Technology of Troyes.

The main requirement for any steganographic tool is to hide data undetectably. Thus, it was important to produce secret images that are not distinguishable from regular images both visually and statistically (more on visual and statistical detection in Appendix 3).

There are many strategies to hide data into images, but in the context of the thesis, two of them were to be employed:

1. **Hiding data into "plain" image.** What is meant by a plain image is a digital image that will not be compressed or modified in any other way than just changing some values in the LSB plane in order to embed the secret message. These images are saved as PNG images.
2. **Hiding data into a JPEG compressed image.** The secret message was to be hidden into the quantized DCT coefficients acquired after completing the lossy part of JPEG compression. Thus, the first steps of JPEG compression had to also be realized in the process of this thesis. The process is explained by Figure 1. These images are saved as JPEG images.

¹ <https://www.mathworks.com/products/matlab.html>

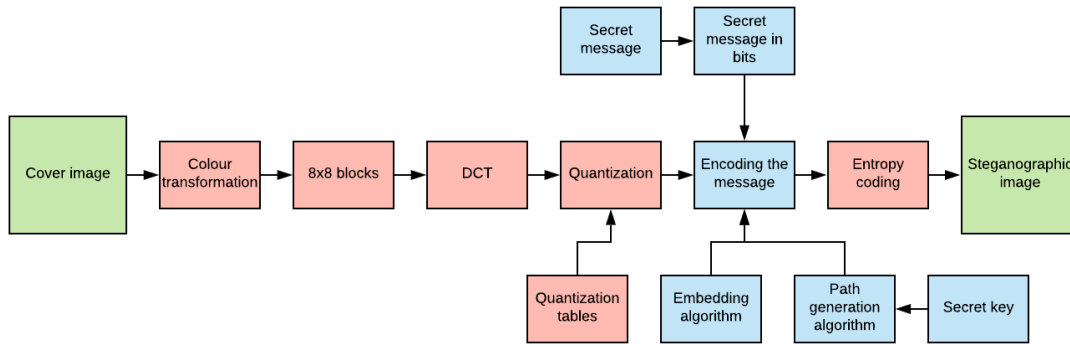


Figure 1. Hiding process of a secret message into a cover image. Blue parts represent the encoding, red parts represent JPEG compression.

Regarding the embedding strategies, the encoder had to use LSB embedding. This choice was made because LSB embedding is one of the more popular and simpler hiding strategies in steganography, while also remaining undetectable [7]. There are many different ways to employ LSB embedding algorithms. In the context of this thesis, it was decided to use LSB matching and LSB replacement. They both modify the LSBs of a cover image, but in different ways (this is thoroughly described in Appendix 2). Thus, both of these embedding strategies could be decoded using the same decoder.

Furthermore, the encoder had to employ at least two path generating algorithms: a "simple" algorithm and an algorithm which generates a pseudo-random path based on a shared secret key. What is meant by a simple algorithm is an algorithm which does not require any kind of additional input from the user and generates the same path for the same image every time. A secret key algorithm will generate the same path for the same image only if the encoder uses the same shared secret key. An additional third algorithm was added, which generates an encrypted path token of a randomized path. The receiver is able to decode the message using the path token.

All of these options were to be parametrizable by the end user, allowing the user to choose the hiding, embedding and path generating strategies to hide the data into the image. For example, the end user can choose to exchange JPEG images using simple zigzagged encoding, or perhaps plain images using a shared secret key where the data is embedded using LSB matching embedding. All of this is needed to provide an application that is useful in research and in academical context or where the end user wishes to have a higher degree of liberation regarding the hiding strategy.

4 Realization

In this chapter, the realization of the thesis and the development of Stegotote will be described. It will cover the topics of technical decisions, realizing the JPEG compression, embedding algorithms, path generating algorithms, steganography application and the user interface. The realizations are described in the chronological order of their implementation.

4.1 Technical decisions

As the area of steganography is quite wide, the scope of this thesis focuses on hiding info inside plain (PNG) and JPEG compressed images. JPEG compression requires many scientific calculations and image manipulations. To make these activities easier, some scientific libraries were used. In this chapter, the most essential technologies and libraries that were used are described.

The thesis was written in **Python 3**¹ programming language. Python is simple in its syntax and very flexible. It also has many libraries to use for scientific calculations and image manipulation.

The external packages and libraries were managed with the **Anaconda**² platform. Anaconda is an extremely resourceful tool to manage scientific libraries and packages for Python.

The most essential library for this thesis is **NumPy**³. When working with images, essentially what is being worked with are multi-dimensional arrays. Greyscale images

¹ <https://www.python.org/>

² <https://www.anaconda.com/>

³ <https://www.numpy.org/>

are 2-dimensional, colour images 3 dimensional arrays. That is why NumPy is needed: it allows powerful N-dimensional array manipulations.

In order to save the quantized DCT coefficients as JPEG images, **Pysteg's Jpeg**¹ package is used. It is a package whose main functionality is implemented in C², but who offers a class called "jpeg" to access the C functionalities in Python code.

4.2 Realization of JPEG compression

The first milestone that was set in the beginning of starting the thesis was implementing the JPEG compression. JPEG compression consists of 5 general steps, which are discussed in further detail in Appendix 1. Additionally, each equation brought out in this chapter is explained further in Appendix 1. The implementation of JPEG compression consists only of the lossy part of JPEG compression, implementing the lossless compression was not in the interest of this thesis.

First of all, the colour space of the image had to be transformed. If the image is grayscale, this process is not needed. But for colour images, it meant splitting the image into its red, green and blue channels. This transforms a 3-dimensional array into three 2-dimensional arrays. Then the channels were transformed into YC_bC_r colour space using Equation (1).

$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} + \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.5 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1)$$

As the main aim of this thesis was not to provide the most optimal compression rate, then no subsampling was done and 4:4:4 subsampling was used.

After transforming the colour space, the compression algorithm was applied to each channel. If the image was greyscale, then only the luminance channel was compressed. For colour images, the Y, C_b and C_r channels were all compressed separately.

¹ <http://www.ifs.schaathun.net/pysteg/pysteg.jpeg.html#>

² [https://en.wikipedia.org/wiki/C_\(programming_language\)](https://en.wikipedia.org/wiki/C_(programming_language))

The compression algorithm consisted of looping through 8×8 blocks of the image, first DCT transforming them and then quantizing the block values. For the DCT transformation, SciPy library's Discrete Fourier Transforms package¹ was used. When the `scipy.fftpack.dct` function is parametrized with the 2nd type of DCT, it will use Equation (2) on the block.

$$d[k, l] = \sum_{i,j=0}^7 \frac{w[k]w[l]}{4} \cos \frac{\pi}{16} k(2i+1) \cos \frac{\pi}{16} l(2j+1) B[i, j] \quad (2)$$

Then, the pre-calculated quantization matrixes were used to quantize the block values according to Equation (3).

$$D[k, l] = \text{round} \left(\frac{d[k, l]}{Q[k, l]} \right), k, l \in \{0, \dots, 7\} \quad (3)$$

For luminance channels, a special luminance matrix is used and for chroma channels, a chrominance matrix is used. On Figure 2, an example of quantized DCT coefficients can be seen. The non-zero values are concentrated into the upper-left corner. The data will be embedded into these values.

```
[[ 43. -38.  -2.  -2.  -0.  -0.   0.   0.]
 [ 27.   7.  -2.   0.   0.  -0.   0.  -0.]
 [ -1.   2.  -0.  -0.   0.   0.   0.  -0.]
 [  2.   1.  -0.   0.  -0.   0.  -0.   0.]
 [ -0.   0.  -0.   0.  -0.  -0.  -0.   0.]
 [  1.   0.  -0.  -0.   0.   0.  -0.  -0.]
 [ -0.   0.   0.   0.   0.   0.  -0.  -0.]
 [  0.   0.  -0.  -0.  -0.  -0.   0.   0.]]
```

Figure 2. Example of quantized DCT coefficients.

After the compression algorithm is applied on all the channels, they are joined back together to form a 3-dimensional array. This is where compression ends in the context of this thesis, as implementing the full JPEG compression is not in the interest of this thesis. At the end of the JPEG compression process, the quantized DCT coefficients are ready to have data embedded into them.

¹ <https://docs.scipy.org/doc/scipy-0.14.0/reference/fftpack.html>

4.3 Realization of path generation

Another prerequisite for hiding data in images was to generate the paths where to hide the secret data. A path is a permutation of the image pixel coordinates (3-dimensional for colour images and 2-dimensional for greyscale images) from which the message can be either hidden or read. A colour image can hold a lot more data than a greyscale image, as instead of having one colour plane there are three (R, G and B for plain PNG images and Y, C_b and C_r for JPEG compressed images). Each coordinate refers to one (unique) pixel in the cover image. As this thesis employs LSB embedding, then each pixel can hold 1 bit of data embedded in its LSB. Thus, a path has to be at least as long as the message decoded into binary.

In this thesis, three different ways to generate paths are used:

1. **Generating the "simple" way.** The simple algorithms always produce the same path from the same input.
2. **Generating from a shared secret key.** The secret key algorithms generate the path based on a secret key value. Thus, the same key always produces the same path on the same image.
3. **Generating a path encrypted with the shared secret key.** The encrypted path algorithms generate a completely random path that is encrypted with the shared secret key and then the encrypted path token is sent to the communicating partner.

The algorithms for plain PNG images and JPEG compressed images are different, as the first works with pixel values and the latter with quantized DCT coefficient values. It is important to be noted that it is possible to hide data into every pixel value of plain images, while it is possible to hide only into the non-zero values of the quantized DCT coefficients. This is because embedding data into zero value coefficients causes visual distortions. Thus, in total of six path generating algorithms were conceived for this thesis. Each of them will be described briefly below.

4.3.1 Generating a simple path for a plain image.

This algorithm generates a path of coordinates in the lexicographical order: from left-to-right, from up-to-down, for each channel.

It is useful when the communicating partners want to communicate without exchanging any secret keys, as the only argument this algorithm takes is the cover image. This algorithm always generates the same path from the same image, as it only depends on the image's dimensions.

The downside is that all of the modifications are happening close together and could be potentially easily noticeable when steganalysed. Also, the path is not calculated based on the message length and will generate a path with all of the coordinates represented, from which the receiver has to identify himself where the secret message ends and the noise begins.

4.3.2 Generating a simple path for a JPEG image.

This algorithm generates a path of coordinates in the zig-zag order. The zig-zag algorithm is the same that is used in JPEG compression and can be seen in Appendix 1 on Figure 18. The only difference with JPEG's zig-zag algorithm, is that instead of arranging all of the coefficients into a one-dimensional array, it only arranges the non-zero coefficients. This is because embedding data into 0 value coefficients causes visual distortions. Thus, coordinates with these values are inherently removed from the path.

Just like with generating a simple path for a plain image, it only requires the cover image to generate the path and it always generates the same path for the same cover image. The difference is though, that the simple path for a JPEG image algorithm does not generate the same path for images with the same dimensions, as the DCT coefficient values depend on the image's pixel values.

Alas, just like with the simple generation for a plain image, it could also be potentially easily detectable and also the path length does not depend on the length of the message. Also, as only non-zero coefficients can be used to hide data, the maximum possible message length is greatly reduced when comparing it to hiding into a plain image. However, this should guarantee being more resistant to detection.

4.3.3 Generating a path with a shared key for a plain image.

This algorithm generates a path of coordinates in a random order based on a shared secret key. Both of the communicating partners can then hide and read the data using the key they have exchanged. The key has to be generated using the Fernet¹ library. Fernet is an implementation of symmetric (also known as “secret key”) authenticated cryptography. This functionality is provided by the application and doesn't have to be done separately. The key is processed and seeded into NumPy's random shuffling function in order to create a random permutation of all the coordinates. By seeding the shuffling method, it is guaranteed to produce the same result for the same key every time.

As this algorithm uses a secret key, it is more complex than the simple generating methods. But it produces a better and less noticeable result, as the data is hidden in a random order in all areas and channels of the image. Thus, it is not so easily detectable as data in only one area.

Alas, it requires for the partners to exchange a key at least once during the communication, which could arise suspicion. In this case, the author proposes to exchange the key using one of the simple algorithms and embedding the key as a secret message, then afterwards using path generation with the shared secret key. Also, this algorithm doesn't take into account the length of the message when generating the path, so again it is up to the communicating partners to identify the end of the message and beginning of noise.

4.3.4 Generating a path with a shared key for a JPEG image

This algorithm generates a path of coordinates of non-zero DCT coefficients in a random order, based on a shared secret key. As with the secret key algorithm for a plain image, the key is generated using the Fernet library and has to be shared between the communicating partners. This algorithm uses the simple path for a JPEG image algorithm to generate the non-zero coefficients and then seeds NumPy's random shuffling method with the key to rearrange them in a random order. The seeding

¹ <https://cryptography.io/en/latest/fernet/>

guarantees that the permutation of coordinates is always the same for the same key and image.

As with the secret key path generation method for a plain image, it is not so easily detectable as the simple method, as the pixels are chosen in a random manner. Instead of using the R, G and B channels to hide the data, it uses the Y, C_b and C_r channels. This allows for a very uniform distribution over the cover image.

In regard to the downsides of this method, it also requires a key to be exchanged at least once to use this communication method. The author proposes the same solution as for the shared key path generation for a plain image (exchanging the key using the simple method). Again, it is up to the reader to distinguish where the message ends and the noise begins when reading the message. Also, as only the non-zero coefficients are usable for hiding data, this method can carry less data as in a plain image, but should be less detectable.

4.3.5 Generating a path encrypted with a secret key for a plain image

This algorithm is different from the previous ones, as the path is only generated once when encoding the message (instead of generating both while encoding and decoding as done with the previous methods). The strategy employed in this method generates a random permutation of coordinates using the Secrets¹ library. The Secrets library generates cryptographically strong random numbers. It is specifically geared towards security and cryptography. After generating the path, it will be encrypted with the shared secret key, creating a token. This token is stored as a text file. The encryption is done using the Fernet library's encrypt method. The token file needs to be sent to the communicating partner alongside with the cover image. Then, on the reader side, the path just needs to be decrypted using the shared secret key. The path will not be generated again.

The downside of this method is that it is very noticeable to send an encrypted text file alongside the cover image on every communication. This problem can be evaded by sending the encrypted path token inside a cover image, and the message in either the same or another cover image, just like when sending the shared secret key.

¹ <https://docs.python.org/3/library/secrets.html>

A difference from the previous methods is the fact that the encrypted path methods take into consideration the length of the secret message when generating the path. Thus, the path will be only as long as the message and will not contain any noise. This is the easiest to read on the receiving end. A high degree of randomness is guaranteed with this method, as it uses the powerful Secrets library.

4.3.6 Generating a path encrypted with a secret key for a JPEG image

This method is very similar to the previous one, while differing on the fact that the data can only be hidden in non-zero coefficients. Thus, it is not as easy as just picking random coordinates from all the planes. This method will choose a random coordinate and check if its DCT coefficient's value is zero or not. If it is zero, it will continue looking. If the value is non-zero, it will add it to the path and move forward to the next message bit. The random generation is also done with the Secrets library. When the path for hiding has been generated, it will again be encrypted and saved as a token, which has to be sent to the communicating partner.

The message hidden with this method should be less detectable, as it is harder to detect bits hidden in JPEG compressed images and the high degree of randomness should ensure that there are no noticeable patterns or areas of modified values. Also, the recovered message is free of any noise and contains only the secret message.

Alas, the path token needs to be sent with every communication. This problem can be overcome in the same manner as described for the encrypted path method for plain images.

4.4 Realization of LSB embedding

There are many different LSB embedding strategies. In this thesis, LSB matching and LSB replacement are used. Their working principles are described thoroughly in Appendix 2. Although they differ in the way they modify the values to match the message, the algorithms always require these three inputs:

1. **Cover image's matrix.** The cover image is the steganography cover object in the context of this thesis. The embedding algorithms take either the plain image's

matrix of the pixel values or the quantized DCT coefficients of the JPEG compressed image.

2. **Message.** The message is the secret message to be hidden into the cover image. It needs to be converted to a string of bits before hiding it. This is done using the `bitarray`¹ module, which allows for easy conversion between bytes (text) and bits.
3. **Path.** The path is a permutation of coordinates of the cover image which signify where to hide the data. The path can be generated in three different kinds of ways, as described in Chapter 4.3. The path generation methods are different for pixel values and DCT coefficient values.

The main principle is the same for every LSB embedding algorithm. They iterate over the given path, check the LSB value of the cover image on this coordinate, and modify it when it does not match. The pseudo-code for both LSB replacement and LSB matching algorithms can be found in Appendix 2.

Alas, the LSB embedding algorithm is dependent on the format of the cover image. This means that the algorithm for embedding into quantized DCT coefficients is not the same as embedding into RGB pixel values. The DCT coefficient value cannot be changed to zero. Thus, additional checks have to be done to prevent this scenario. When modifying the RGB plane, the value cannot be more than 255 or less than 0. Again, these cases have to be prevented by checking the value beforehand. The decoding algorithm stays the same for both plain and JPEG images

In practice, six embedding algorithms were created for the author's tool. Three of them were for colour images with 3-dimensional arrays, three of them for greyscale images with 2-dimensional arrays. Out of the six algorithms, four use LSB matching and two LSB replacement. This is because using LSB replacement with the DCT coefficients is more complicated. Values 0 and 1 cannot be embedded into, as they risk changing the number of non-zero coefficients. Thus, a new decoder that doesn't read 0 or 1 values would have had to be developed. Finally, out of the four algorithms employing LSB

¹ <https://pypi.org/project/bitarray/>

matching, two use the DCT coefficients and two use RGB pixel values. Both of the LSB replacement algorithms use RGB pixel values.

4.5 User interface

Stegote is a command-line tool. There are two ways to enter the necessary information for the tool. Firstly, it can be used either by specifying the flags straight on the command-line. All the possible flags can be seen on Figure 3.

```
Application for hiding messages in images. Bachelor thesis 2019

optional arguments:
  -h, --help                show this help message and exit
  --path PATH                Path of the cover image
  --dest_path DEST_PATH     Destination folder path to save the secret image
  --generate_key             Enter this argument to generate a shared secret key.
  --encode                  Enter this argument to encode a message into an image.
  --decode                  Enter this argument to decode a message from an image.
  --compression             Enter this argument if you want to specify whether you
                             want to hide the message in aJPEG compressed image or
                             not. [y/n] for YES or NO
  --embedding               Enter this argument if you want to specify the
                             embedding strategy. [r/m] for LSB REPLACEMENT of LSB
                             MATCHING.
  --encoding                Enter this argument if you want to specify the
                             encoding. [s/k/p] for SIMPLE encoding, encoding using
                             a shared KEY or encoding using a PATH token encrypted
                             with the shared key.
  --key                     Enter this argument if you need to use a secret key.
                             Enter the key.
  --path_token              Enter this argument if you need to use a path token.
                             Enter the token path.
```

Figure 3. View of the Stegote tool after entering the --help command.

Secondly, Stegote can be used by answering the command prompts presented based on the user's choices. The user has to enter at least whether they wish to encode data into an image, decode the message from an image or generate a secret key. An example of using the tool to encode data into an image can be seen on Figure 4. The input prompts asking the user to specify the manner of hiding can be seen. If the user does not wish to parametrize the hiding method, default values will be used. Lastly, Stegote will always print out the manner of encoding to notify the user of their choices and let the user know where to find the encoded image.


```
[(thesis) Triinus-MBP-2:thesis triinuerik$ python main.py --encode
[pysteg.jpeg] $Id: __init__.py 2204 2011-04-05 11:43:38Z georg $
Input the path of the cover image: /Users/triinuerik/PycharmProjects/thesis/images
/i3.jpg
Input the secret message to hide: very very secret message hidden in an image
Input the path of the directory to save the secret image: /Users/triinuerik
Do you want to encode/decode the message in JPEG compressed image (press Enter for
default)? [y/n] y
Do you want to select the compression quality (press Enter for default)? [0 (lowes
t) - 100 (best)]
Do you want to use SIMPLE encoding, encoding using a shared KEY or encoding using
a PATH tokenencrypted with the shared key? (press Enter for default)? [s/k/p] s
Do you want to use LSB REPLACEMENT embedding or LSB MATCHING embedding? (press Ent
er for default)? [r/m] m
Secret JPEG image encoded with zigzag encoding.
Secret image saved to /Users/triinuerik
```

Figure 4. Example of using the Stegote tool to encode a message.

On Figure 5, the image containing the secret message can be seen. The image was encoded in the same manner as shown on Figure 4. It can be seen that there are no visual distortions.

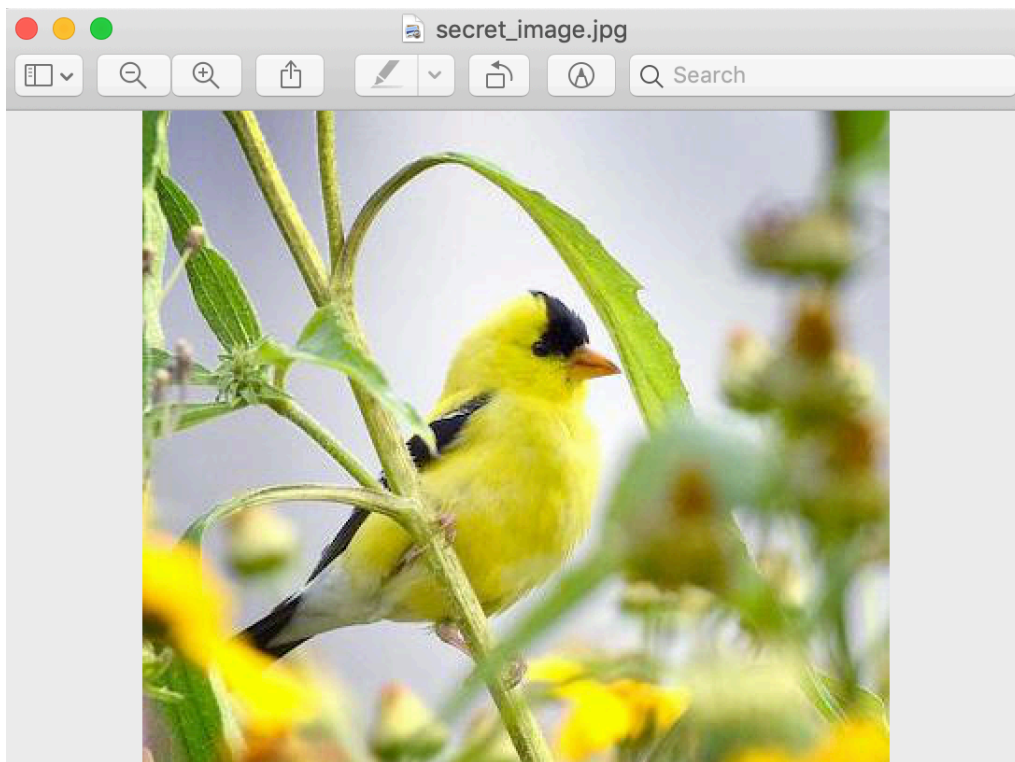


Figure 5. Example of a secret image with data embedded into it.

When decoding a message, the receiving person needs to know the manner in which the data was hidden into the image, except for the embedding because both LSB embedding strategies (LSB matching and LSB replacement) use the same decoder. An example of decoding a message is on Figure 6.

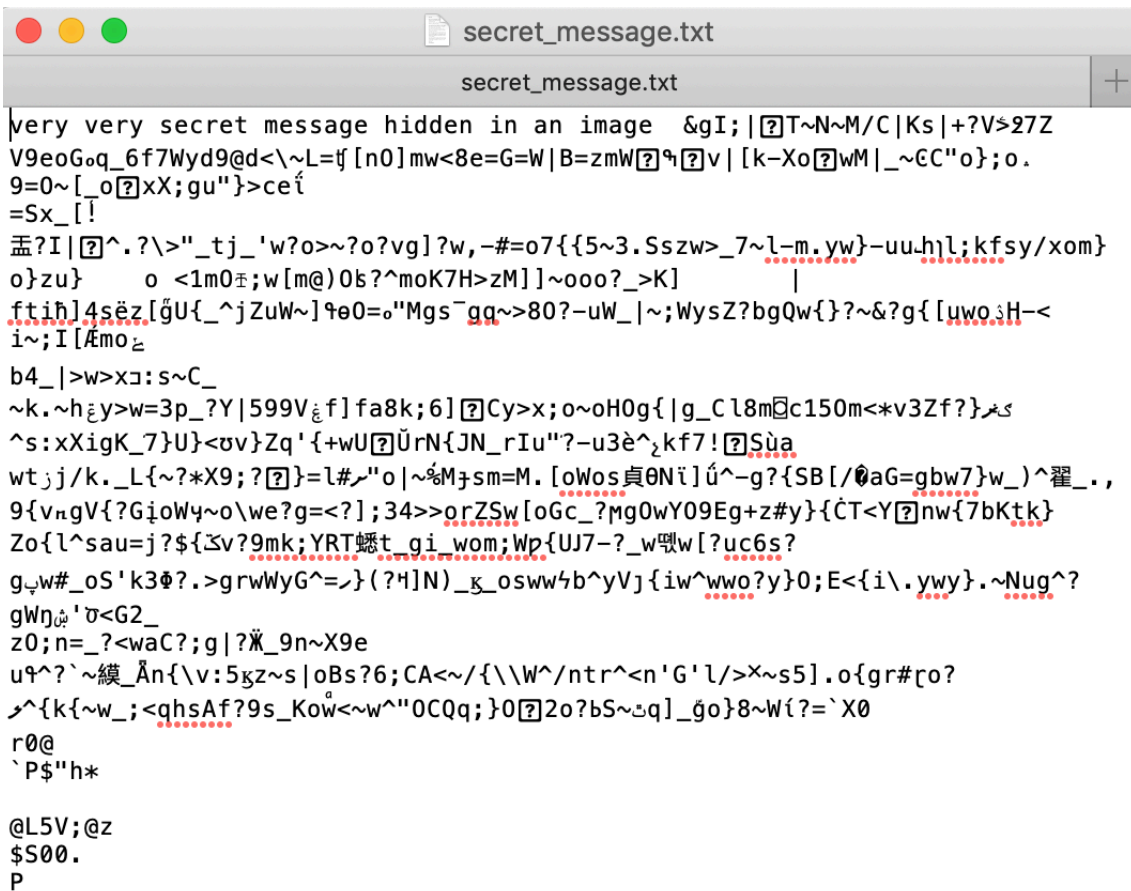
```

(thesis) Triinus-MBP-2:thesis triinuerik$ python main.py --decode
[pysteg.jpeg] $Id: __init__.py 2204 2011-04-05 11:43:38Z georg $
Input the path of the cover image: /Users/triinuerik/secret_image.jpg
Do you want to encode/decode the message in JPEG compressed image (press Enter for
default)? [y/n] y
Do you want to use SIMPLE encoding, encoding using a shared KEY or encoding using
a PATH tokenencrypted with the shared key? (press Enter for default)? [s/k/p] s
Secret message decoded from JPEG image using zigzag decoding.
Secret message can be found in: /Users/triinuerik/secret_message.txt

```

Figure 6. Example of using the Stegote tool to decode a message.

The application will generate a text-file in the same folder as the decoded image containing the secret message. For the simple encoding and encoding based on a secret key, it is up to the user to recognize where the secret message ends and noise begins. This is because the path generation algorithms are not aware of the length of the message, they only generate the same permutation of coordinates. For the encrypted path token encoding, the secret image will be printed instead of saved to a file. An example of a decoded message is on Figure 7.



```

secret_message.txt
secret_message.txt
very very secret message hidden in an image &gI;|T~N~M/C|Ks|+?V>27Z
V9eoGoo_6f7Wyd9@d<\~L=ŧ[n0]mw<8e=G=W|B=zMWŧŧv|[k~XoŧwM|_~CC"o};o.
9=0~[_oŧxX;gu"}>ceí
=Sx_[ŧ
盂?I|ŧ^.?\"_tj_'w?o~?o?vg]?w,-#=o7{{5~3.Sszw>_7~l~m.yw}-uu.h}l;kfsy/xom}
o}zu} o <1m0±;w[m@)0b?^moK7H>zM]]~ooo?_>K] |
ftih]4sëz.[ŧU{_^jZuW~}ŧe0="Mgs_ŧgq~>80?-uW_|~;WysZ?bgQw{}?~&?g{[uwo;H-<
i~;I[ŧmo_
b4_|>w>xɔ:s~C_
~k.~hεy>w=3p_?Y|599Vεf]fa8k;6]ŧCy>x;o~oH0g{|g_Cl8mŧc150m<*v3Zf??}ع
^s:xXigK_7}U}<σv}Zq' {+wUŧŧŧrN{JN_rIu"?-u3è^_kf7!ŧSùa
wt;j/k._L{~?*X9;?ŧ]=l#r"o|~%M}sm=M.[oWos真θNì]ú^-g?{SB[/ŧaG=gbw7}w_)^翟_. ,
9{vngV{?GïoW4~o\we?g=<?};34>>orZSw[oGc_?mg0wY09Eg+z#y}{ŧT<Yŧnw{7bKtk}
Zo{l^sau=j?${zsv?9mk;YRT蟋t_gi_wom;Wp{UJ7-?_wŧw[?uc6s?
g_pw#_oS'k3ϕ?.>grwWyG^=,}{?ŧ]N)_g_oswW4b^yVj{iw^wwo?y}0;E<{i\ywy}.~Nug^?
gWŧ'σ<G2_
z0;n=_?<waC?;g|?ŧ_9n~X9e
uŧ^?~纒_Ān{\v:5kz~s|oBs?6;CA<~/{\W^/ntr^<n'G'l/>x~s5}.o{gr#[o?
ŧ^k{~w_;<qhsAf?9s_Kow<~w^"OCQq; }0ŧ2o?bS~ɔq]_ŧo}8~Wí?=`X0
r0@
`P$"h*

@L5V;@z
$S00.
P

```

Figure 7. Example of a decoded secret message.

In order to hide a message using any other method than the simple method, a shared secret key has to be generated. Figure 8 shows an example of generating a secret key for the user.

```
[(thesis) Triinus-MBP-2:thesis triinuerik$ python main.py --generate_key
[pysteg.jpeg] $Id: __init__.py 2204 2011-04-05 11:43:38Z georg $
Your shared secret key is: 4MAX8fP-hlfY0Zr69Hq0M-wEZuxATvJ_82R1bipIrWk=
Share this key with the person you wish to send messages to. Don't lose it!
```

Figure 8. Example of using the Stegote tool to generate a key.

5 Validation

In this chapter, Stegote is compared with other similar solutions previously discussed in Chapter 2.2 and the images containing secret information hidden with Stegote are steganalysed with a detection tool in order to verify the undetectability a secret message. Additionally, a usability test that was carried out on Stegote to test the tool's UI/UX.

5.1 Comparison with similar solutions

In Chapter 2.2, five major readily available steganography tools were discussed. In this chapter, they are compared with the author's tool, Stegote.

Having analysed these discussed in Chapter 2.2 it is clear that **none of them offer the same degree of parameterisation as Stegote**. The comparison is brought out in Table 1. Some of the other tools only offer different encryption algorithms for encrypting the hidden data, but this does not allow to choose the way the data is hidden in the image.

Table 1. Comparison of five major tools and the author's tool, Stegote.

Tool	Embedding algorithm	Output file	Is parameterizable	Supported OS
OpenStego	LSB	PNG	No	Not dependent on OS
Hide'N'Send	LSB and F5	JPEG	Yes, embedding algorithm	Windows XP/Vista/7
SteganoG	Unknown	BMP	No	Windows 7/8/10
Steghide	Graph-theoretic matching algorithm	JPEG, BMP, WAV and AU	No	Unix and Windows
Jsteg	LSB	JPEG	No	Unix and Windows
Stegote	LSB matching and replacement	JPEG and PNG	Yes, embedding algorithm, path generation algorithm and file format	Not dependent on OS

The only tool that offers some choice is Hide'N'Send, which allow to use either the LSB or F5 embedding. But this tool is only supported on older Windows operating systems, which reduces its availability for users. The tools are usually geared towards one specific hiding strategy, which means that if there is a wish to change the hiding strategy, the tool cannot be used anymore.

Thus, for a user who wishes to easily change their hiding strategy or who wishes to have control over the way the data is hidden, Stegote is the best choice.

5.2 Steganalysis on Stegote

StegExpose¹ is a steganalysis tool developed by Benedikt Boehm that specializes in detecting LSB steganography in lossless images. StegExpose was thus used to test the plain images encoded with Stegote that are saved as PNG files. StegExpose rating algorithm is derived from an intelligent and thoroughly tested combination of pre-existing pixel based steganalysis methods including Sample Pairs by Dumitrescu (2003), RS Analysis by Fridrich (2001), Chi Square Attack by Westfeld (2000) and Primary Sets by Dumitrescu (2002) [4].

Benedikt Boehm tested StegExpose [4] against images created with four different tools, namely LSB-Steganography², OpenPuff³, OpenStego and SilentEye⁴, which all use LSB embedding. In his article [4], Boehm calculated the True Positive Rates (TPR) and False Positive Rates (FPR) of each threshold for the arithmetic mean for all four aforementioned steganalysis methods [4]. From his findings [4], the Receiver Operating Characteristic (ROC) curve seen on Figure 9 was conceived. As mentioned in Appendix 3, a ROC curve describes the performance of a detection or diagnostic tool by plotting the TPRs and FPRs. In Appendix 3, a typical ROC curve of a good detector can be seen. Thus, it can be concluded that StegExpose is efficient in detecting steganography embedded with namely LSB-Steganography, OpenPuff, OpenStego and SilentEye.

¹ <https://github.com/b3dk7/StegExpose>

² <https://github.com/RobinDavid/LSB-Steganography>

³ https://embeddedsdsw.net/OpenPuff_Steganography_Home.html

⁴ <https://silenteye.v1kings.io/>

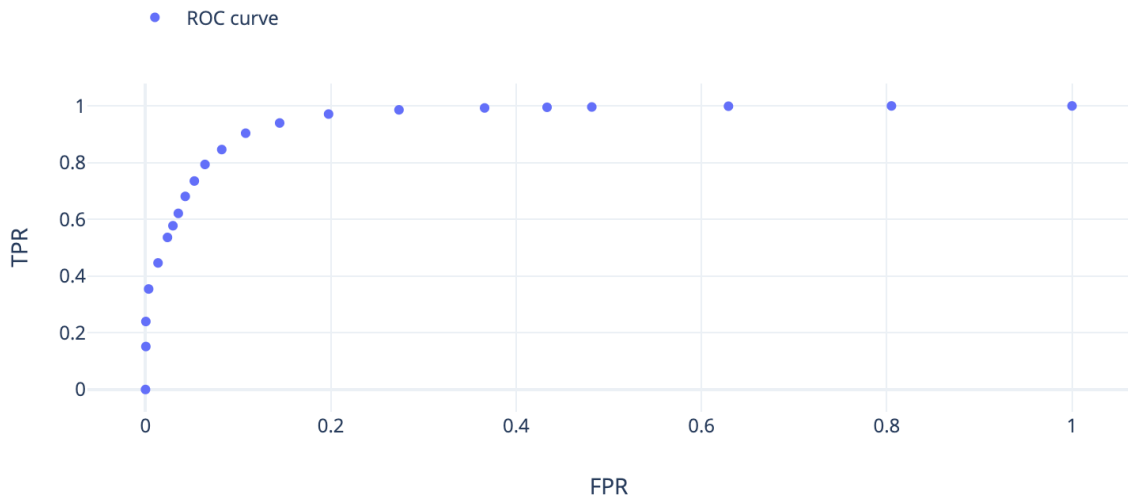


Figure 9. ROC curve of StegExpose tested against LSB-Steganography, OpenPuff, OpenStego and SilentEye.

For the purpose of testing the strength of Stegote, a dataset of 40 PNG files was created. Of the 40 images, 16 are regular unmodified images and 24 are images that have data embedded into them with the author's tool, at least once in every possible combination of the parameters (colour image or greyscale, simple or secret key or encrypted path token path generation, LSB replacement or LSB matching embedding).

StegExpose permits to modify steganography threshold that determines the level at which files are considered to be hiding data or not. By default the threshold is 0.2, as it was determined to be the best trade-off between fall-out (False Positive Rate) and sensitivity (True Positive Rate) [4]. For reducing the number of false negatives (missed detections), it is recommended to set the threshold to ~ 0.15 .

Stegote was first tested against StegExpose at the recommended threshold 0.2, which yielded no detections. All of the regular images were identified as such, but at the same time none of the steganographic images were detected. In order to reduce the number of missed detections, threshold 0.15 was used (as recommended by the manual). Again, the results stayed the same. In fact, no changes happened until threshold ~ 0.08 , where three steganographic images were detected. All of these images used the same cover image, which hints that the cover image had been chosen poorly. As the threshold was decreased, more steganographic images were detected, but also the number of false alarms started to increase. At threshold 0.03, there were four correct detections, but also

two false alarms. The trend of increased number of false alarms accompanying the increased number of correct detections continued for all of the thresholds. Table 2 expresses the true and false positive for some selected cut point thresholds and their TPR and FPR.

Table 2. True and false positives, TPR and FPR for selected thresholds for the StegExpose tool used against the author's tool, Stegote.

Threshold	True positive (Correct detection)	False positive (False alarm)	TPR	FPR
0.2	0 / 24	0 / 16	0	0
0.08	3 / 24	0 / 16	0.125	0
0.05	3 / 24	0 / 16	0.125	0
0.03	4 / 24	2 / 16	0.1667	0.125
0.025	7 / 24	3 / 16	0.2917	0.1875
0.02	8 / 24	5 / 16	0.3333	0.3125
0.015	9 / 24	7 / 16	0.3750	0.4375
0.01	9 / 24	9 / 16	0.3750	0.5625
0.0085	14 / 24	9 / 16	0.5833	0.5625
0.007	17 / 24	11 / 16	0.7083	0.6875
0.005	19 / 24	12 / 16	0.7917	0.75
0.003	24 / 24	16 / 16	1	1

By plotting the TPR and FPR against each other, the ROC curve of the StegExpose tool against the author's tool is achieved. In Appendix 3, some examples of good and bad ROC curves are given. The more the ROC curve resembles a linear line, the worse the detector is at detecting the hidden message. A linear line expresses detection as good as a random guess. As seen on Figure 10, the ROC curve of StegExpose tested against Stegote resembles a linear line. This means that StegExpose is not able to effectively detect steganography hidden with the Stegote

These results suggest that the **steganographic methods used in the author's tool are not detectable.**

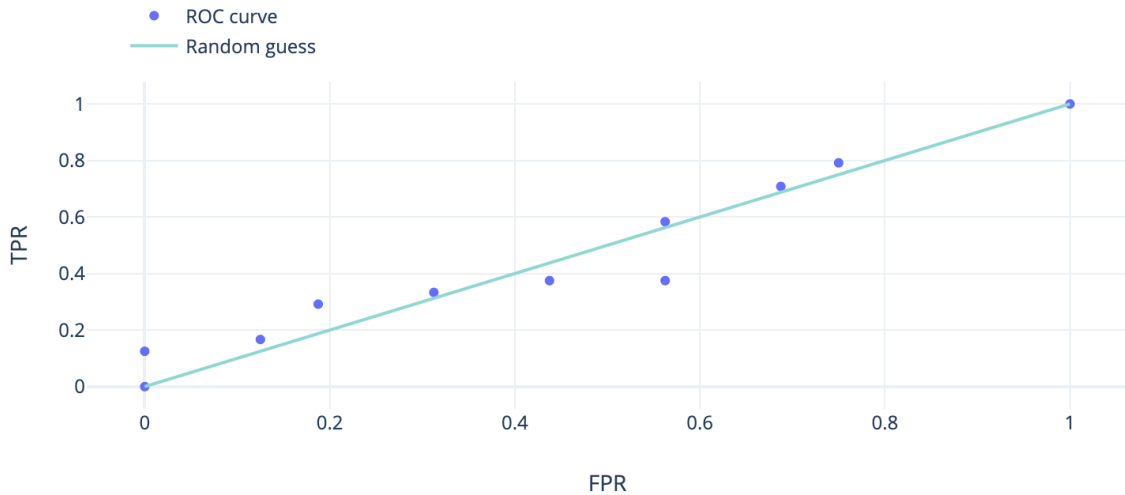


Figure 10. ROC curve of StegExpose tested against Stegote.

5.3 Usability testing

The ISO 9241-11 standard [17] officially defines usability as "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". The Interaction Design Foundation lists [18] the three main goals of a usable interface as:

1. Being easy for the user to become familiar with and competent in
2. Being easy for users to achieve their objective
3. Being easy to recall the user interface and how to use it on subsequent visits

In order to test the user interface (UI) and user experience (UX) of Stegote, a brief usability test was carried out. The test was carried out on three people who could be likely users of a tool like Stegote. They all had a background in info technology and had used the command-line before but were not proficient in it. Before beginning the test, the users were explained what Stegote does and how image steganography is possible. They were asked to carry out three tasks (see in Appendix 4). Each task asked the user to hide a message of their choice into a specified image in a specified manner. After encoding the message, they were asked to decode it. Each task asked the user to hide the message in a different manner. While the users were solving the tasks, the author acted as a silent observer, only answering questions or helping the user along when they were confused.

All three users found it hard to understand what to do in the beginning. As they were not proficient in using the command-line, they did not know that the "--help" flag displays all the possible commands to enter. But after pointing out the command needed to enter for encoding and decoding, they found it easy to use from that point on. All three found that after completing the first task, the next two were easier and more intuitive to follow.

The first user mentioned positively the input prompts Stegote gives, saying that "they are easy to follow". The user was confused by some word choices, namely about the "shared secret key" and proposed to use just "secret key". Overall, the user found the tool very interesting and regarded it positively.

The second user had difficulty using the tool because they do not use a MacBook and was thus having some trouble copy-pasting the file path and finding the saved pictures. Even though it seemed confusing, they said "everything you need to do, you are told to do" in reference to the fact that it was not very difficult to use. The second user also found some word choices of the input prompts confusing, namely when asked to enter the desired file format and encoding method. Overall, they liked the tool.

Before testing the third user, the author created a quick guide on the Github page of Stegote, where the basic commands were brought out next to screenshots. This was very helpful as the user had a point of reference of which commands to enter. Again, the biggest obstacle was using a MacBook. Overall, the user carried out the tasks with no big difficulties.

In conclusion, **all three users regarded the usability of Stegote positively**, bringing out the main difficulties as not being very familiar with the command-line or the operating system. Aside from these factors, the users carried out the tasks with no big difficulties. All three goals listed by the Interaction Design Foundation [18] were generally fulfilled.

Their mentioned recommendations were taken into account and the proposed fixes were made to Stegote's UI.

6 Limitations and future work

It was intended to use 10 different ways to hide data into images, but one of them, hiding data into a JPEG image with the shared secret key, continued to fail. The error is not coming from the author's code, but rather from Pysteg's Jpeg package. When saving and reading again from the JPEG file, the amount of non-zero coefficients changed slightly every time, which suggests an error in the package's saving functionality. This does not allow to generate the same random permutation with the same secret key, as the lengths of the arrays were always slightly different. The Jpeg package appears to be very experimental and is not well-documented, which made finding the bug difficult. Alas, the method is tested and works flawlessly on the DCT coefficient level on both encoding and decoding, so if the bug in the Jpeg package gets fixed, it is possible to get the 10th hiding option to work.

In the future, an obvious area of improvement is adding even more ways to hide data into images. The main improvement could be done in the area of embedding. Even though LSB embedding remains undetectable in many cases, it is one of the most researched area of steganography. The author proposes to add either alternative embedding strategies and / or some state of the art LSB embedding methods like adaptive LSB embedding or LSB rotation. Additionally, the application could benefit from a Graphical User Interface (GUI) to make it more intuitive and easier to use for people who are not familiar with command-line tools.

7 Conclusion

The goal of this thesis was to create a customizable steganography tool that allows users to have a high degree of choice in the way their data is hidden. The tool had to hide data into digital images in an undetectable manner. These goals were fulfilled.

Stegote enables users to hide data into plain PNG and JPEG compressed images, using three different kinds of path generation algorithms and two different LSB embedding strategies, LSB replacement and LSB matching. The tool offers a simple command-line interface.

According to comparative analysis to similar tools, Stegote offered much more flexibility regarding the hiding strategies.

Stegote was tested against a steganalysis tool [4], which was not able to detect the steganographic images any better than a random guess.

A brief usability test was carried out on Stegote, where users regarded Stegote's UI/UX in a generally positive manner.

References

- [1] J. Fridrich, *Steganography in Digital Media: Principles, Algorithms and Applications*, New York: Cambridge University Press, 2010.
- [2] A. Jeeva, V. Palanisamy and K. Kanagaram, "Comparative Analysis of Performance Efficiency and Security Measures of Some Encryption Algorithms," *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 3, pp. 3033-3037, 2012.
- [3] "Examining The Importance Of Steganography Information Technology Essay," UKEssays, 2018.
- [4] B. Boehm, "StegExpose - A Tool for Detecting LSB Steganography," School of Computing University of Kent, England, 2014.
- [5] D. Frith, "Steganography approaches, options, and implications," *Network Security*, vol. 2007, no. 8, pp. 4-7, 2007.
- [6] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1079 - 1107, 1999.
- [7] R. Sharma, R. Ganotra, S. Dhall and S. Gupta, "Performance Comparison of Steganography Techniques," *International Journal of Computer Network and Information Security*, vol. 10, no. 9, 2018.
- [8] E. Walia, P. Jain and N. Navdeep, "An Analysis of LSB & DCT based Steganography," *Global Journal of Computer Science and Technology*, 2010.
- [9] M. Celik, G. Sharma, A. Tekalp and E. Saber, "Lossless generalized-LSB data embedding," *IEEE Transactions on Image Processing*, vol. 14, no. 2, 2005.
- [10] M. Maes, "Twin Peaks: The Histogram Attack to Fixed Depth Image Watermarks," in *International Workshop on Information Hiding*, 1998.
- [11] J. Bierbrauer and J. Fridrich, "Constructing good covering codes for applications in steganography," *Transactions on data hiding and multimedia security III*, 2008.
- [12] P. Malathi and T. Gireeshkumar, "Relating the embedding efficiency of LSB Steganography techniques in Spatial and Transform domains," *Procedia Computer Science*, September 2016.
- [13] QianMao, "A fast algorithm for matrix embedding steganography," *Digital Signal Processing Volume*, vol. 25, pp. 248-254, 2014.
- [14] S. Sugathan, "An improved LSB embedding technique for image steganography," in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, Bangalore, 2016.
- [15] R. A. Subong, A. C. Fajardo and Y. J. Kim, "LSB Rotation and Inversion Scoring Approach to Image Steganography," in *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Nakhonpathom, 2018.

- [16] S. De Vuono, "Github," 25 October 2013. [Online]. Available: <https://github.com/StefanoDeVuono/steghide/blob/master/doc/steghide.1>. [Accessed 12 July 2019].
- [17] International Organization for Standardization, "ISO 9241-11:2018, Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts".
- [18] P. Morville, "Usability," Interaction Design Foundation, [Online]. Available: <https://www.interaction-design.org/literature/topics/usability>. [Accessed 7 August 2019].
- [19] M. Rabbani and P. W. Jones, "Digital Image Compression Techniques," SPIE Press, Bellingham, 1991.
- [20] P. J. Kostelec, "Taking Advantage of Spatial Redundancy," [Online]. Available: <https://www.cs.dartmouth.edu/~geelong/spatial/spatialRedundacy.html>. [Accessed 17 April 2019].
- [21] "Compression," Umeå University, 2005. [Online]. Available: <https://www8.cs.umu.se/kurser/TDBC30/VT05/material/lecture8.pdf>. [Accessed 9 May 2019].
- [22] "Human visual system model," [Online]. Available: https://en.wikipedia.org/wiki/Human_visual_system_model. [Accessed 9 May 2019].
- [23] KeyCDN, "Lossy vs Lossless Compression," KeyCDN, 21 November 2018. [Online]. Available: <https://www.keycdn.com/support/lossy-vs-lossless>. [Accessed 18 April 2019].
- [24] J. Janet, D. Mohandass and S. Meenalosini, "Lossless Compression Techniques for Medical Images In Telemedicine," 16 March 2011. [Online]. Available: <https://www.intechopen.com/books/advances-in-telemedicine-technologies-enabling-factors-and-scenarios/lossless-compression-techniques-for-medical-images-in-telemedicine>. [Accessed 19 July 2019].
- [25] W3Techs, "Usage statistics of JPEG for websites," [Online]. Available: <https://w3techs.com/technologies/details/im-jpeg/all/all>. [Accessed 18 July 2019].
- [26] G. K. Wallace, "The JPEG Still Picture Compression Standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, February 1992.
- [27] "Discrete cosine transform," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Discrete_cosine_transform. [Accessed 2 May 2019].
- [28] J. Liu and J. Wang, "JPEG Compression and Ethernet Communication on an FPGA," [Online]. Available: https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2009/jl1589_jbw48/jl1589_jbw48/index.html. [Accessed 9 May 2019].
- [29] FileFormat.info, "Run-Length Encoding (RLE)," [Online]. Available: https://www.fileformat.info/mirror/egff/ch09_03.htm. [Accessed 19 July 2019].
- [30] M. Sharma, "Compression Using Huffman Coding," *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, no. 5, 2010.
- [31] H. Wang and S. Wang, "Cyber Warfare: Steganography vs. Steganalysis," *Communications of the ACM*, vol. 47, no. 10, October 2004.
- [32] "Wikipedia," 7 May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Sensitivity_and_specificity. [Accessed 17 July 2019].

- [33] S. H. Park, J. M. Goo and C.-H. Jo, "Receiver Operating Characteristic (ROC) Curve: Practical Review for Radiologists," *Korean J Radiol*, March 2004.
- [34] C. Peters, "Wikipedia," 6 February 2011. [Online]. Available: <https://en.wikipedia.org/wiki/Talk%3AYCbCr>. [Accessed 2 May 2019].
- [35] Spears & Munsil, "Choosing a Colour Space," [Online]. Available: <http://spearsandmunsil.com/portfolio-item/choosing-a-color-space/>. [Accessed 2 May 2019].

Appendix 1 – Compression

This chapter focuses on image compression: what it is, why it is needed, the problems it solves and how it is done. Also, it describes one type of image compression, JPEG compression. JPEG compression is one of the most widely used compression methods, as it achieves to reduce the size of images considerably, without causing noticeable visual distortions. JPEG compression is used in the scope of the practical part of this thesis to hide information into JPEG images.

Image compression

The vast majority of images we encounter are compressed using one of the many compression standards created. In this section it will be discussed why this is so and what are the benefits of image compression.

Why is image compression needed?

By the beginning of the 90s, digital imaging had taken a huge leap in advancement. For the first time in history, different types of media could be easily converted into digital form. But during the early years of image digitalization, there was a big problem: the vast amount of data needed to represent a raw digital image.

As an example, let's consider a low-resolution colour image for TV quality. Assuming the resolution is 512 x 512 pixels/colour, with each pixel encoded by 8 bits, and 3 colours (RGB), then the total size of one image reaches approximately 6×10^6 bits [19].

The large file sizes combined with the slow transmission speeds back then meant that it was almost impossible to apply digital images realistically. Taking into account the typical transmission speed of a telephone line (9600 bit/s), it meant that the aforementioned image would take around 11 minutes to transmit [19].

These figures show the difficulty of storing and transmitting one low-resolution image. When taking a look at a digitalized 35mm negative photograph, the size increases

tenfold [19]. Storing any kind of high-resolution, specialized or professional images would prove close to impossible to store, especially small hard drive sizes back then.

That is why the question of image compression became prevalent. Even though technology has advanced since the 90s, these problems still remain actual and image compression is still widely used.

How is image compression possible?

Image compression relies on the fact that digital images contain quite a fair amount of redundancy [19]. It means that digital images tend to have excessive amount of information. Images usually have similar qualities, which allows to optimize how they are represented. These redundancies can be roughly divided into three categories:

1. **Spatial redundancy**, meaning that pixels located near each other have the tendency to have similar values. In essence, it is presumed that an image will have larger areas of pixels in similar intensities and with similar values. This leads to possible prediction of the neighbouring pixel values [20].
2. **Spectral redundancy**, meaning the correlation between different colour planes [19]. Colour planes are the different components that form the representation of an image, e. g. in an RGB image we have red, green and blue colour planes.
3. **Temporal redundancy**, meaning that in the case of receiving multiple images in sequence (e.g. a video broadcast) the pixels tend to more or less keep a value similar to the previous image [19].

Image compression is based on trying to remove or lessen these three redundancies. In essence, it is unnecessary for each pixel to carry a lot of information and the behaviour of pixels in images is in many cases predictable.

As an example, it is easy to imagine a portrait of a person [20]. On the portrait there would be larger areas of pixels with similar colours/luminosity: a lighter area for the face and skin, maybe a darker area of pixels representing the clothes, etc. It is unlikely for a dark pixel to appear in the middle of the person's face, as seen on Figure 11 – it is possible to predict relatively well that a large number of the pixels composing the face have similar values.



Figure 11. Example: a portrait [20] where some pixels have been changed to carry unlikely values, i.e. dark pixels in the middle of a face and vice versa.

Psychovisual interpretation

In essence, raw digital images contain a lot of information that the human eye either does not see or does not notice big changes to. A human's visual perception differs from a camera's. The eye of a camera will catch a wide variety of colours and nuances that a human eye will never or hardly notice. This principle of redundancy is the basis for compressing images.

Psychovisual redundancy comes from the fact that the human eye does not respond with equal intensity to all visual information presented [21]. A human will not analyse the separate pixels that make up an image. Instead, an observer searches for distinct features and tries to find recognizable objects [21]. To simplify the behaviour of this complex system, the Human Visual System (HVS) model was created. In the HVS model, the different areas of biology and psychology are gathered in order to clarify the visual processes that are not yet fully known.

Some assumptions the HVS model has are, for example, that the human eye is more susceptible to high contrast, has low colour resolution and is more sensitive to motion [22]. In addition, the human mind has a very strong face recognition system. In the case of the Hollow-Face illusion, facial recognition rules over depth perception. This means that instead of seeing an inverted and hollow mask, the human eye will instead perceive it as a face.

The HVS model is taken advantage of in JPEG compression. According to the HVS model, changes to details in higher frequency are not as perceptible as in lower frequency [22]. Thus, these components can be compressed more without causing too severe visual distortions. This principle is used while performing the DCT transform (explained in Appendix 2).

Lossy and lossless compression

There are countless algorithms created to take advantage of redundancy in images. These compression methods could be categorized into two groups:

1. **Lossless compression**, where the reconstructed file is identical to the original image [19]. By looking at each bit's value, it would not have changed from the original value. This means that lossless compression is completely reversible.
2. **Lossy compression**, where the compressed file has suffered distortions and the reconstructed image is not identical to the original file. That means, some data from the original file is lost [23]. Although, these distortions might not be visually noticeable and might not be perceived by the eye under regular viewing conditions [19].

Although ideally lossless compression is preferred, sometimes the reduced size of the compressed file is not enough. With lossless compression, the integrity of the image is well preserved, but the compressed file could still be too big. This might not be a problem for some use cases, when only a few files are stored, there is a lot of storage space available, etc. Using lossless compression is common with medical, graphical or technical images [24].

In ordinary life, high preservation of the image quality is not necessary and reduced file size has a lot more importance. Thus, lossy compression is widely used, as for many daily use cases visually equal images serve well enough. As seen on Figure 12, lossy compressed files are not visually different, but are much smaller in size. But on closer inspection, as on Figure 13, severe visual distortions can be seen. Lossy compression serves well enough for photographs.



Figure 12. Although they seem almost identical, the image on the right is ~80% smaller than the image on the left.



Figure 13. When closely looked, the compressed image (right) has highly visible distortions compared to the original image (left).

JPEG compression

JPEG is an acronym of Joint Photographic Experts Group (JPEG), who developed the first international digital image compression standard in 1992. This standard is still widely used nowadays and is one of the most popular standards [25]. It was meant to be a general-purpose compression standard to fit the needs of the majority of still-image applications [26].

The idea behind JPEG compression relies on the fact that people perceive images differently than computers: not as a collection of pixels as matrixes but a collection of segments filled with texture [1]. Thus the JPEG compression standard aims for high compression rate with "very good" or "excellent" visual fidelity [4], which means that JPEG compression is a lossy method that aims to not have any visually perceptible disruptions. Additionally, the compression rate is parameterizable, so the user could specify a rate that corresponds to their needs.

JPEG compression consists of five steps, which will be described by the following sections.

Colour transformation

In this step, the colour of the image is changed from the RGB model to the $YCrCb$ model.

The RGB colour model comes from the fact that the human eye has three different receptors – cones – in the eye retina. These cones are receptive to red, green and blue colour. These cones send electrical signals to the human brain, where the signal is perceived as a colour. This additive nature of the RGB model can be witnessed on Figure 14.

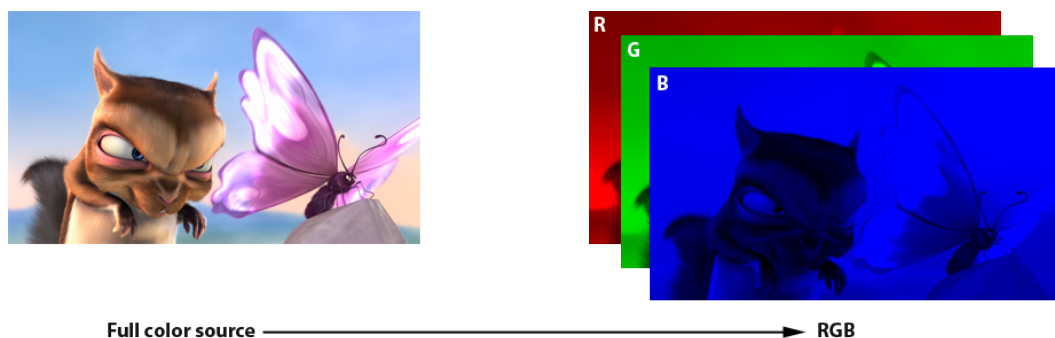


Figure 14. An image divided to its red, green and blue components [21].

The RGB model is taken advantage of in hardware displays, where colour is produced by combining three values from the RGB vector. For example LED screens are made up of red, green and blue light emitting diodes, which in group of threes produce all the visible colours a human eye can see.

Even though the RGB model describes perceivable colours well, it carries redundant information because the three signals are highly correlated between themselves [1]. That means, it is not the most economical in the way it carries information. For this, the $YCrCb$ model was created.

$YCrCb$ model takes advantage of the fact that biologically, human eyes are much less sensitive to changes in chrominance than to luminance. This means that our eyes notice changes in brightness/darkness more than equal changes in colour. The $YCrCb$ colour space consists of 3 axes: luminance, red chrominance and blue chrominance. This can be witnessed on Figure 15.

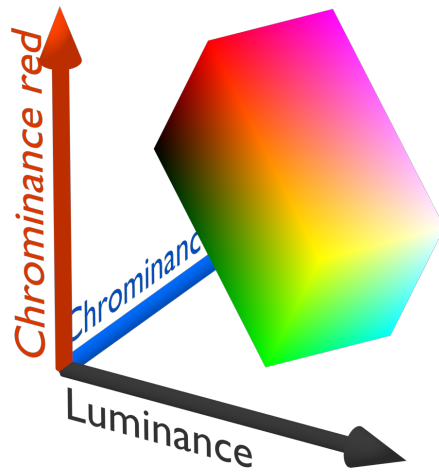


Figure 15. Visual representation [22] of the $YCrCb$ model.

The $YCrCb$ colour model is obtained by linearly transforming the RGB components using Equation (1).

$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} + \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.5 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1)$$

The luminance Y is defined as a weighted linear combination of the RGB channels determined by the sensitivity of the human eye to the red, green and blue colours [1]. To adjust all three components to the same range representable by 8 bits, the chrominance components will be added 128, so they also would fall into the $\{0, \dots, 255\}$ range.

The resulting $YCrCb$ components will then divide into one black-and-white channel accompanied by two chroma channels, as seen on Figure 16.

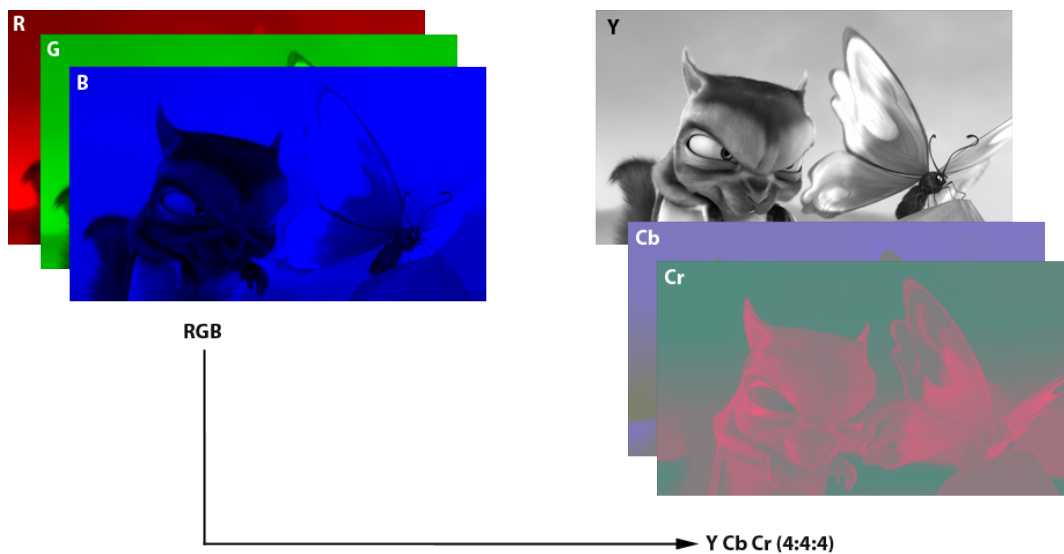


Figure 16. RGB to YCrCb transformation visualized [21], presuming no subsampling has been done.

Division into blocks and subsampling

In this step, the Y , C_r and C_b signals are divided into blocks. The chrominance signals might be further subsampled before block division [1].

As the DCT transformation and quantization steps are performed on 8×8 matrixes, it is necessary to first divide the image into corresponding blocks of pixels. The luminance signal Y is always divided into blocks of 8×8 pixels, as the human eye is much more sensitive to changes in luminance and it is needed to retain all information about this signal. C_r and C_b channels, on the other hand, can be subsampled to achieve a higher compression rate.

The image will be divided into 16×16 pixel macroblocks, which each can yield 1, 2 or 4 blocks for each chrominance, depending on the subsampling type. If the macroblock is subsampled by a factor of 2 in each direction, each macroblock will only have one 8×8 pixel C_r block and one 8×8 pixel C_b block. This nation is usually abbreviated as $4 : 1 : 1$ [1]. If the C_r and C_b blocks are subsampled only along one direction, the macroblock will yield 2 chrominance blocks for each, abbreviated as $4 : 2 : 2$. If no subsampling is

done, the notion would be 4 : 4 : 4 [1]. Before DCT transforming the blocks, all pixel values will have 128 subtracted from them.

DCT transform

The Discrete Cosine Transform (DCT) will transform each block's $Y_C r C_b$ signals from the spatial domain to the frequency domain [1]. The DCT can be interpreted as a change of basis for the 8×8 pixel matrixes. DCT is a Fourier-related transform similar to the Discrete Fourier Transform (DFT) but using only real numbers [27].

For an 8×8 pixel block of values $B[i, j]$, $i, j = 0, \dots, 7$, the 8×8 block of DCT coefficients $d[k, l]$, $k, l = 0, \dots, 7$ is computed as a linear combination of values,

$$d[k, l] = \sum_{i,j=0}^7 \frac{w[k]w[l]}{4} \cos \frac{\pi}{16} k(2i+1) \cos \frac{\pi}{16} l(2j+1) B[i, j] \quad (2)$$

where $w[0] = \frac{1}{\sqrt{2}}$, $w[k > 0] = 1$ [1]. The coefficient $d[0, 0]$ is called the DC coefficient while the remaining coefficients with $k + l > 0$ are called the AC coefficients [1]. The results of a DCT transform represent the spatial frequency information of the original block at discrete frequencies corresponding to the index into the matrix [28].

The spatial frequency representation of DCT can be seen on Figure 17. It is clear that the top-left elements have lower frequencies, while the bottom-right elements have higher frequencies [28]. Most of the original information can be reconstructed from the lower frequency coefficients which is due to the high-energy compaction in those coefficients [28]. Moreover, the human eye is less perceptive to errors regarding the high-frequency elements [28]. Considering these factors, it is clear that when there are errors in the lower frequency components, they will be more noticeable to the human eye.



Figure 17. The spatial frequency representation of DCT [24].

The Discrete Cosine Transform is invertible, which is important for decompressing JPEG images. The IDTC is

$$B[i, j] = \sum_{k, l=0}^7 \frac{w[k]w[l]}{4} \cos \frac{\pi}{16} k(2i + 1) \cos \frac{\pi}{16} l(2j + 1) d[k, l] \quad (4)$$

Quantization

In this step, the resulting matrix of the DCT transform is divided by a quantization matrix and the results are rounded to the nearest integer value. The quantization matrix consists of integer values and it is also called the quantization step.

The purpose of quantization is to enable representation of DCT coefficients using fewer bits [1]. This leads to loss of information, which means this is the lossy part of JPEG compression. During quantization, the DCT coefficients $d[k, l]$ are divided by quantization steps from the quantization matrix $Q[k, l]$ and rounded to integers [1]

$$D[k, l] = \text{round} \left(\frac{d[k, l]}{Q[k, l]} \right), k, l \in \{0, \dots, 7\} \quad (3)$$

The larger the quantization step, the fewer bits can be allocated to each DCT coefficient and the larger the loss of information [1]. This leads to visual distortions in images.

The standard and most widely used quantization matrix for the luminance component is the 50% quality standard JPEG quantization matrix,

$$Q_{50}^{(\text{lum})} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \quad (5)$$

For the chrominance components, a special chrominance matrix similar to the luminance one is used.

Encoding and lossless compression

In this step, the quantized DCT coefficients are arranged in the zig-zag order, encoded using bits and then losslessly compressed [1].

The zig-zag arrangement means organizing the elements of the matrix from the top-left corner in a serpentine manner, as seen on Figure 18. This will turn a 8×8 matrix into a 1x64 array, where the non-zero values will be concentrated to the beginning of the array and non-zero values to the end. In essence, the zig-zag algorithm will concentrate similar values next to each other.

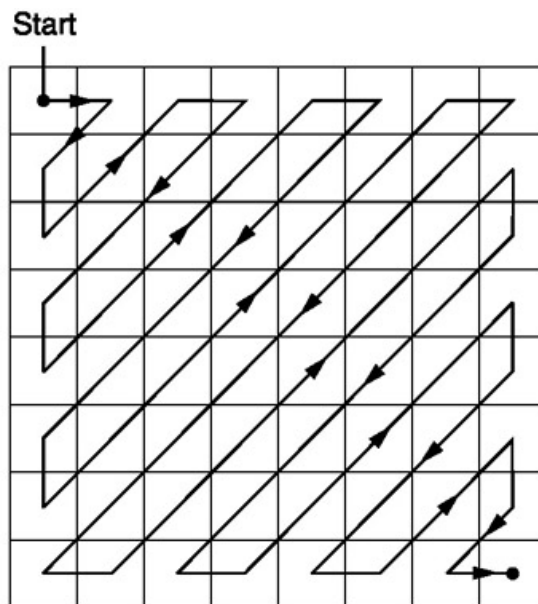


Figure 18. The visual presentation [24] of the zig-zag algorithm on an 8×8 block.

After completing the zig-zag algorithm, the DC and AC components are encoded. For the DC components it is common to use delta encoding. Delta encoding means

comparing the DC components of the current and previous blocks and storing the difference between them.

For the AC components, Run Length Encoding [29] is used. The RLE method transforms an array of values into their corresponding values to sequence pairs. The first element in the pair represents the value while the second element represents the number of consecutive values. As an example, for a string "aaaabbccddd", the encoded result would be "a4b2c1d3"

The final step before packing the image is Huffman coding [30]. This method looks at the frequency of each appearing symbol and chooses the optimal bit code for each of them. The principle is that more frequently appearing symbols will be represented using fewer bits. Huffman coding is a type of lossless compression, which means that no losses to the image quality will happen at this stage.

Appendix 2 – Steganography

Steganography is the practice of communicating secret messages using a cover object. The main goal of steganography is to conceal the fact that any secret communication is happening at all. That is why it is useful to use a cover object, because photos, songs, videos, etc. are sent every day between people all over the world. Sending a photo will not rouse suspicion, as it is a completely ordinary thing to do. Thus, in the context of this thesis, digital images have the main focus.

Steganographic system

In order to have a successful steganographic communication, the communicating partners need to agree to a protocol that they will follow. The five key elements of the protocol are described in this chapter. On Figure 19 the visualization of these elements and their interactions can be seen.

Source of covers. A cover object needs to be something that the communicating partners would be exchanging if they were not having a secret communication. It has to be something ordinary, which is why photos are ideal for steganography. The typical attributes of a digital image cover source would be the image format, origin, resolution, typical content type, etc [1]. It also needs to be able to fit a reasonable amount of secret data in it, i.e. have a sufficient capacity.

Data embedding and data extraction algorithms. The embedding and extraction algorithms can be thought as the most important parts of the steganographic system [1], as they determine the secrecy and the quality of the sent message. Through the process of embedding a message, the sender determines the image (or another object) that communicates the secret message [1]. The extraction algorithm will uncover the hidden data communicated by the sender. The embedding and extraction algorithms often depend on the steganographic key, which determines the way the data is hidden and how it can be later recovered.

Source of the steganographic keys. Steganographic keys are the rules that determine how data is embedded into and extracted from the cover object. The selection of the steganographic key often includes a randomized parameter. This means that a reasonable strategy would be to select the steganographic key randomly, with uniform distribution, from the set of all possible keys [1].

Source of messages. The secret message that is communicated needs to be an optimal size. The shorter the communicated message, the lower is the probability that it will be detected. But in that case the number of distinct transfers needed to be made to communicate a message with a significant meaning is larger. On the other hand, by embedding as much information into an image as the embedding algorithm allows, the less instances of communication need to be made. But this carries a higher risk of detection. Thus, the communicating partners need to keep in mind that the message source has a major influence on the security of the steganographic system [1].

Channel for exchanging data. The channel is the physical way where the sent data is transferred from the sender to the receiver, for example the Internet. In the context of steganography, it is assumed that the channel is monitored by a warden. The warden can assume a passive role, which means monitoring the traffic but not interfering with it. The warden can also assume an active role, where he can attempt to disrupt the communication by compressing, cropping or otherwise distorting the images. Lastly, the warden can assume a malicious role, where he will try to crack the steganographic method and impersonate the communicating partners and send messages himself or otherwise confuse the communicating partners [1].

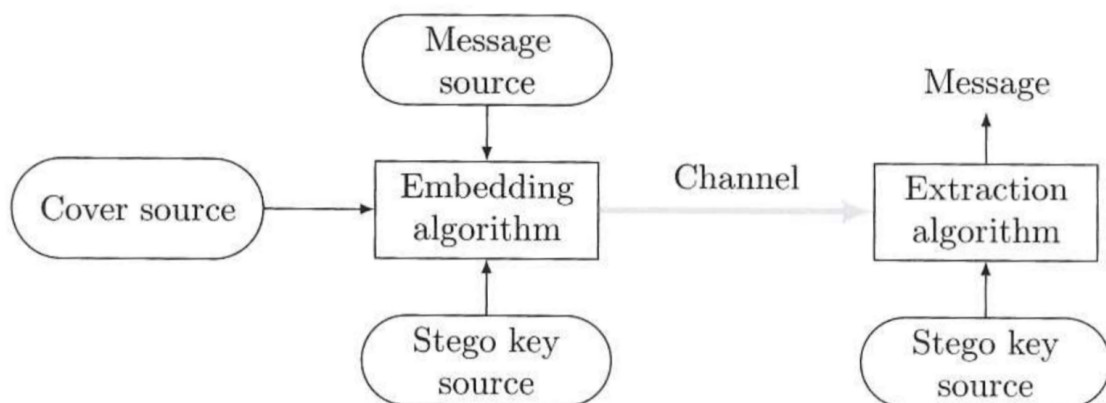


Figure 19. Visualisation of the elements of a steganographic system.

To sum up these key elements of the steganographic system, Dr. Jessica Fridrich formulates the problem of steganography as

"... finding embedding and extraction algorithms for a given cover source that enable communication of reasonably large messages without introducing any embedding artefacts that could be detected by the warden. In other words, the goal is to embed secret messages undetectably" [1].

Steganography paradigms

The embedding and extraction algorithms are the most important part of a steganographic system. Generally, three main architectures of steganographic algorithms are recognized. These paradigms determine the internal mechanism of the embedding and extraction algorithms [1]. The three fundamental architectures are:

1. **Steganography by cover selection.** This method allows the partners to communicate by choosing a cover image from a fixed database that already has the hidden message inside [1]. It means that the partners have before-hand decided the meanings of different pictures, e. g. a picture in portrait mode means "yes" and in landscape mode it means "no". It has the advantage of security, as the images are not modified in any way, but it is difficult to communicate a meaningful message with this method.
2. **Steganography by cover synthesis.** A sender using this method constructs himself an object that conveys a message [1]. He can take a photo of himself wearing specific clothes, being in a specific location, etc., with all of those details conveying a special meaning.
3. **Steganography by cover modification.** This method is the most widely used method and it is the mainstream approach to steganography. A secret message is hidden by making modifications to the cover image in order to embed secret information [1]. As it is the paradigm that is used in the context of this thesis, it will be discussed more thoroughly in the next chapter.

Steganography by cover modification

Steganography by content modification is the most studied paradigm today. It is so widely used that it has become synonymous with the term "steganography" itself. In this chapter, the basic definitions and concepts of this paradigm will be discussed.

The principle of steganography by cover modification is that the sending partner will make modifications to a cover image in order to embed secret data [1]. In order to do this, the communicating partners use

$C \dots \text{set of all cover objects } x \in C$ (6)

$K(x) \dots \text{set of all steganographic keys for } x$ (7)

$M(x) \dots \text{set of all messages that can be communicated in } x$ (8)

A steganographic scheme is a pair of embedding and extraction functions Emb and Ext,

$\text{Emb} : C \times K \times M \rightarrow C$ (9)

$\text{Ext} : C \times K \rightarrow M$ (10)

such that for all $x \in C$ and all $k \in K(x)$, $m \in M(x)$,

$\text{Ext} (\text{Emb} (x, k, m), k) = m$ (11)

To give a brief explanation, the Equation (11) demonstrates the nature of steganography by content modification. Equation (11) is also visualized on Figure 20. In order to send a secret message, a secret message m is embedded into a cover object x in a manner determined by the shared steganographic key k , using the embedding function Emb. In other words, steganographic image y will be $y = \text{Emb} (x, k, m)$. The sender will transfer this image y to the receiver over a channel, who will extract the secret message m from the image y with the help of the shared steganographic key k , using the extraction function Ext. In other words, the message is extracted as $m = \text{Ext} (y, k)$. It can be clearly seen that only by knowing the shared key k , it is possible to communicate regardless of the message or the cover object itself. It also demonstrates the invertible nature of the embedding and extracting functions, i.e. the steganographic scheme.

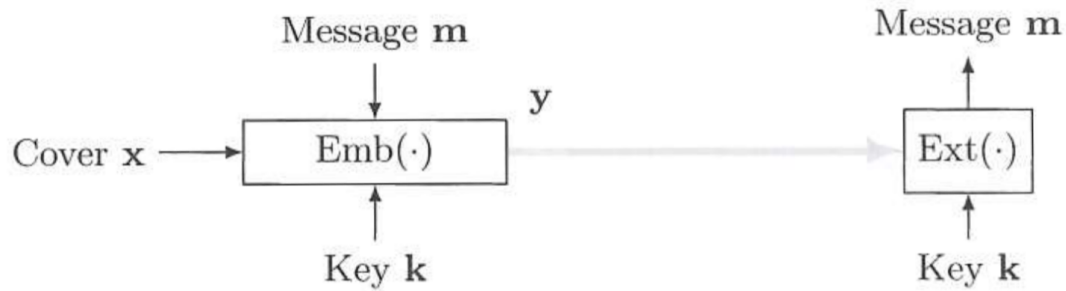


Figure 20. Visualization of steganography by cover modification.

The number of messages that can be communicated in a cover object x depends on the steganographic scheme and on the cover object itself [1]. These two concepts of how much information it is possible to embed in a cover object are known as

1. **Average embedding capacity.** One way to think of the capacity of a cover image is how many bits it is overall possible to embed in an image. For a grayscale 512×512 image, the maximum length of the secret message can be $M = \{0, 1\}^{512 \times 512}$. Thus, the definition of the embedding capacity in bits is $\log_2 |M(x)|$.
2. **Relative embedding capacity.** As most images are compressed using some standard or another, the content of the image has to be taken into account. For a JPEG compressed image, the number of bits it is possible to embed depends on the number of non-zero DCT coefficients. Thus, we arrive at a relative embedding capacity of $\frac{\log_2 |M(x)|}{n}$, where n is the number of elements that could possibly be used for embedding.

These two ways to measure an image's capacity are quite theoretical. While they give a good foundation on the overall embedding capacity of an image, in practice the more commonly used concept is the **steganographic capacity**. An image's steganographic capacity is defined as the maximum number of bits that can be embedded without introducing detectable artefacts [1]. An image's steganographic capacity is typically much smaller than its embedding capacity [1]. This concept is also known as the secure payload. Unfortunately, determining the secure payload of an image is quite difficult, as it is heavily dependent on the individual image, specific steganographic image and even the channel of communication.

LSB embedding

Least Significant Bit (LSB) embedding can be considered as the simplest and most common steganographic algorithm type. It follows the steganography by cover modification paradigm. It can be applied to any collection of numerical data represented in digital form [1]. The two LSB embedding algorithms used in the frame of this thesis are LSB replacement and LSB matching.

Both LSB replacement and LSB matching modify the LSBs of the cover image. LSB embedding is quite liberal in its usage. It can be employed for all kinds of mediums, both images or sound files. In the frame of this thesis it works with both pixel values and quantized DCT coefficients of the cover image.

LSB embedding works by taking the binary representation of either the pixel or coefficient in big-endian form, where the most significant bit is first, and modifying the last bit. This last bit is the LSB, whose significance regarding the whole binary value is the smallest. The decoding algorithm for both LSB replacement and LSB matching is the same.

LSB replacement

LSB replacement (also known as LSB substitution or LSB flipping) is a type of LSB embedding algorithm. It is one of the most popular embedding algorithms used and is often used synonymously with LSB embedding. The pseudo-code of LSB replacement is shown on Figure 21.

```
for each Coordinate in Path:
    if LSB of CoverImage[Coordinate] does not equal MessageBit:
        LSB of CoverImage[Coordinate] = FlipLSB
    else:
        continue
    MessageBit = NextMessageBit
return CoverImage
```

Figure 21. Pseudo-code of LSB replacement.

LSB replacement's main principle is to check along a previously-generated shared path if the LSBs of the cover image are the same as the message bits. If it is not the same, it "flips" the LSB: 0 becomes 1 and 1 becomes 0. It means that if the algorithm is

expecting a 0, but it finds a 1, it will flip the bit's value while disregarding how it changes the pixel's or coefficient's value as a whole.

The downside of LSB replacement is that it creates problems due to its asymmetry [1]. It means that even values are never decreased and odd values are never increased during embedding. That leaves it vulnerable to detection. This leads to the LSB matching algorithm, which uses symmetrical embedding.

LSB matching

LSB matching (also known as ± 1 embedding) is a type of LSB embedding algorithm. It uses the same principle for embedding as LSB replacement, as it changes the LSB to match the message bit. But instead of blindly flipping the bit value, it randomly increases or decreases it. Thus, with LSB matching, the other bits of the pixel/coefficient may also be modified as the LSB increases or decreases. In the most extreme case, even all of the bits of the pixel/coefficient could be modified, for example when the value $127_{10} = 01111111_2$ is increased and changes to $128_{10} = 10000000_2$. The pseudo-code of LSB matching is shown on Figure 22.

```
for each Coordinate in Path:
    if LSB of CoverImage[Coordinate] does not equal MessageBit:
        LSB of CoverImage[Coordinate] += Random[1, -1]
    else:
        continue
    MessageBit = NextMessageBit
return CoverImage
```

Figure 22. Pseudo-code of LSB matching.

In practice, the LSB matching algorithm is not that simple. The exact algorithm depends on the cover image, where additional checks have to be done for edge cases. When embedding a message into the pixel values of an image, the value 255 can only be decreased and 0 only increased. When embedding a message into the DCT coefficients, it has to be checked that a coefficient is not changed to 0. Thus, the value 1 can only be increased and value -1 only decreased.

Decoding LSB embedded messages

As mentioned before, the decoding algorithm for LSB replacement and LSB matching are the same. This comes from the fact that they are both LSB embedding type

algorithms and the message can be recovered from the LSBs. The decoding algorithm can be seen on Figure 23.

```
for each Coordinate in Path:  
    MessageBits += LSB of CoverImage[Coordinate]  
return MessageBits
```

Figure 23. Pseudo-code of decoding LSB embedded message.

To recover the hidden message of the receiver's side, the decoder has to just read long the previously-generated shared path and add all of the LSBs together to form the complete message in bits.

Appendix 3 – Steganalysis

Steganalysis is the activity of trying to detect steganography. Steganalysis is the complementary action to steganography. As mentioned in Appendix 2, in a steganographic system we expect that the communication channel is being monitored by a warden, who tries to detect any kind of embedding. This warden is, in fact, performing steganalysis.

It is not the main goal of steganalysis to crack the message hidden in the steganographic object. As the goal of steganography is to embed secret messages undetectably, the warden only needs to become suspicious of some kind of embedding in order to perform successful steganalysis. Successful steganalysis, also called a successful attack, is when the steganalysist is able to distinguish between cover objects and steganographic objects with a probability better than random guessing [1].

As the focus of this thesis is on digital images, then this chapter will also concentrate on that domain. Generally, there are two kinds of steganalysis techniques [31]:

1. **Visual steganalysis**, which tries to reveal the presence of secret communication through inspection, either with the naked eye or with the assistance of a computer [31]. Naked eye steganalysis is possible when the cover image is smooth or the message was inserted into an area of the image that is smooth. In that case, the distortion of the pixels is more visible. Computer-assisted steganalysis can mean, for example, extracting the LSBs of the image and trying to detect any kind of unusual properties in the LSB plane.
2. **Statistical steganalysis**, which tries to reveal tiny alterations in an image's statistical behaviour caused by steganographic embedding [31]. Statistical steganalysis techniques are usually aimed at specific embedding algorithms, as each of them changes the cover image in their own way. General purpose steganalysis tools do not perform as well as targeted techniques.

Generally, there exist 4 kinds of predictions that a detection tool can make [32]. In the context of steganalysis in this thesis, they are:

1. **True positive**, which means that the steganographically modified image was correctly identified as such. In other words, it describes a correct identification.
2. **False positive**, also known as a false alarm, is a prediction where a regular image was identified as a steganographic image. In other words, it describes an incorrect identification.
3. **True negative**, which means that a regular unmodified image was correctly identified as such. In other words, it describes a correct rejection.
4. **False negative**, also known as a missed detection, is a prediction where a steganographically modified image was identified as a regular image. In other words, it describes the incorrect rejections.

ROC curve

Sensitivity and specificity, which are defined as the number of true positive decisions divided by the number of actually positive cases and the number of true negative decisions divided by the number of actually negative cases, respectively, constitute the basics of measuring the performance of any kind of diagnostic tests [33]. When the results of a test fall into one of two obviously defined categories, such as either the presence or absence of steganography, then the test has only one pair of sensitivity and specificity values [33]. However, in many situations, making a decision in a binary mode is both difficult and impractical [33]. This is why the Receiver Operating Characteristic (ROC) curve becomes useful. The ROC curve describes the performance of any kind of detection or diagnostic tool. The curve plots two parameters: True Positive Rate (TPR) and False Positive Rate (FPR). TPR, also known as sensitivity, measures the percentage of steganographic images that are correctly detected out of all the steganographic images. It is calculated in the following way:

$$\text{TPR} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \quad (12)$$

FPR expresses the probability of a false alarm. It is calculated so:

$$\text{FPR} = \frac{\text{false positives}}{\text{false positives} + \text{true negatives}} \quad (13)$$

On Figure 24 (a), a general example of how a ROC curve looks like can be seen. The X-axis represents FPR and Y-axis TPR. Their values are always between 0 and 1. The linear line expresses a random guess. Thus, the more curved towards the upper-left corner the ROC curve is, the better is its detection rate. On Figure 24 (b), a ROC curve of a not very efficient tool can be seen, because the curve resembles a linear line. On the contrary, on Figure 24 (c) it is possible to see the ROC curve of a very performant steganalysis tool. When comparing two curves it is found that they intersect, as on Figure 24 (d), it is hard to decide which is better than the other.

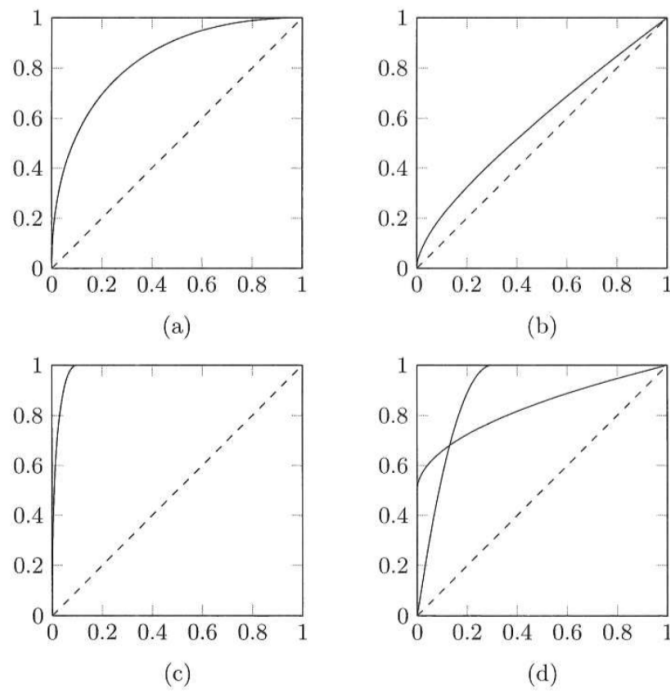


Figure 24. Examples [20] of ROC curves.

Appendix 4 – Usability testing tasks

1. Encode the picture: `/Users/triinuerik/PycharmProjects/thesis/images/i3.jpg` with a secret message of your choice.

You can save it to folder: `/Users/triinuerik/test`

Use JPEG compression with simple encoding and LSB replacement.

Decode the picture and find the text file containing the secret message.

2. Generate a secret key and save it somewhere.

Encode the picture: `/Users/triinuerik/PycharmProjects/thesis/images/i5.jpg` with a secret message of your choice.

You can save it to folder: `/Users/triinuerik/test`

Encode the message without JPEG compression, use secret key encoding and LSB matching.

Decode the picture and find the text file containing the secret message.

3. Encode the picture: `/Users/triinuerik/PycharmProjects/thesis/images/i1.jpg` with a secret message of your choice.

You can save it to folder: `/Users/triinuerik/test`

Use JPEG compression and path token encoding, using the secret key from the previous use case (choose whichever LSB embedding method).

Decode the picture using the path token that was generated during the encoding.