UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Chair of Practical Computer Science IV:
Dependable Systems Engineering

Bachelor's Thesis

Implementation and Visualization of Steganography Procedures in CrypTool2

as part of the degree program Bachelor of Science Business Informatics
submitted by

Sally Addad
Matriculation number 1606277

on September 21, 2020.

Supervisors:	Prof. Dr. Frederik Armknecht
	Prof. Dr. Bernhard Esslinger
	Dr. Nils Kopal

1 Abstract

Steganography is an information hiding procedure using ordinary non-secret cover files. Various techniques and algorithms were developed over the time to hide the data in text files, images, audio and video files, etc.

The goal of this thesis is to offer an interactive visualization of the Least Significant Bit (LSB) and Bit Plane Complexity Segmentation (BPCS) image steganography techniques, and to implement simple text steganography techniques using zero-width spaces, capital letters, or dots added under or above the letters using UTF8 encoding.

This thesis introduces the above mentioned techniques and discusses the plugins implemented in CrypTool 2 (CT2) in addition to how to use them and experiment with the plugins' different modes and configurations. Furthermore, a comparison between these different techniques in regard to their hiding capacity, perceptual transparency and resistance to steganalysis is provided along with the advantages and limitations of each technique.

Contents

List of Figures

Acronyms

BPCS	Bit Plane Complexity Segmentation
CGC	Canonical Gray Coding
CT2	CrypTool 2
GIF	Graphics Interchange Format
GUI	Graphical User Interface
JPEG	Joint Photographic Experts Group
LSB	Least Significant Bit
MSB	Most Significant Bit
PBC	Pure Binary Coding
PNG	Portable Network Graphics
RGB	Red Green Blue
UTF8	8-Bit Universal Character Set Transformation Format

2 Introduction

Throughout history, people have been in search of ways to communicate secretly. This could have religious, political, military or just private motives [20, Page 1]. This is especially the case when communicating over the Internet. Large amounts of data get transmitted on a daily basis, and cryptography is applied to achieve data confidentiality. Cryptography renders the communication incomprehensible to unwanted third parties. In special cases, it is even desired to conceal the whole fact that secret data is being communicated, here is where steganography comes in place.

Steganography comes from the Greek words steganós (covered) and -graphy (writing or drawing). Steganography defines the procedure of hiding information in an ordinary non-secret medium (e.g., text, image, and audio file) [23].

2.1 Motivation

Although analog steganographic techniques were widely used for a long time, computer-based steganography first came into focus when an encryption ban was considered. This ban was intended to avoid the encryption of illegal communication and illegal actions on the Internet [20, Chapter 17]. If that ban were to be implemented, steganography would remain the only option to communicate secretly using ordinary not encrypted files [20, Chapter 17]. This need resulted in many techniques that implement digital steganography.

Nowadays, cryptography and steganography can be combined to ensure strong secrecy of sensitive information in special situations [9]. A good practice would be to encrypt the classified data and then embed it in the cover medium. In the event the secret message is detected and extracted from the cover file, the message would still be encrypted and the attack attempt would still fail [28].

Moreover, understanding and anticipating steganography is also valuable when defending the security of a system against malware. Many malware writers make use of steganographic methods to hide malicious payloads (Stegomalware). Steganography is for the attackers' purpose much more secure than just using obfuscated code [25].

2.2 Goals and Objectives

The goal of this thesis is to implement some of the different image and text steganography techniques in CrypTool 2 and discuss them. Following are the CT2 components that I built in the scope of this thesis to help achieve this:

- Image Steganography: conceals an input secret message in a given cover image using LSB or BPCS algorithm. The new LSB mode is an enhanced version where the user is able to choose other bits to hide the message and not only the LSBs.

- Text Steganography: conceals a secret message in a cover text using zero-width spaces, capital letters or marking letters in the cover text with special characters.

These components offer a visualization of each technique. They also focus on didactics and experimentation as oppose to solely implementing steganographic techniques.

2.3 Related Work

Many software applications and online tools implement different interfaces to hide information, examples include:

- Components already implemented in CT2 for steganography. Among these are components for Permutation, LSB and Insertion steganography. Offered are also templates to demonstrate the encryption and decryption using these techniques.

- A number of GitHub repositories implementing LSB and BPCS image steganography. Inspiration for my BPCS implementation came from the BPCS-Steg-Java GitHub project [10].

- Unicode Text Steganography Encoders/Decoders [27] and other websites offer useful interfaces to hide secret messages in text using Unicode characters, white spaces etc. Some websites require an input cover text, other generate random cover texts which could look like a spam email as implemented on the spammimic website [22].

The components I implemented for this thesis complement these projects by adding an experimental and visual element. This way the user can understand visually what is happening in each process while testing different configurations and analyzing the result in a detailed way.

There are many other tools and websites that offer different types of steganography as well. However the mentioned examples are the most relevant and related to the objectives of this thesis.

3 Fundamentals

This chapter provides some background information about steganography and introduces CrypTool 2 (CT2). Important basic concepts and fundamentals regarding steganography are discussed. This includes comparing the various types of steganography, explaining the components of a steganography model and defining the criteria of a good steganographic technique. Lastly, the term steganalysis is briefly introduced with some of its basic approaches.

3.1 History

The first steganographic techniques go back to ancient Greece, methods back then included carving a secret message in wood panels and then covering them with a layer of wax. Recipients who knew about this procedure would then find the message successfully after removing the layer of wax. Simultaneously, during the transport process, the panel seemed as a normal blank panel to other people who came across it [4].

Other approaches included using invisible ink, text semagrams, musical notes, special symbols and much more creative methods founded based on available resources and the situation where the secret communication was needed.

Steganography was used to a great extent by spies in the times of war as well in order to convey confidential military information about attacks and locations to their homeland. An example would be Velvalee Dickinson who was a Japanese spy in the second world war, she would send innocent looking letters that were supposedly about puppet orders, and hidden in these letters was the classified information that would help the Japanese during the war attacks [20, Chapter 1].

3.2 CrypTool 2

CT2 is the modern successor of CrypTool 1, a well-known e-learning platform for cryptography and cryptanalysis [1]. It offers an interface to create workflows by adding and combining different elements (plugins) in a project. A user can either drag implemented components (for input, output and cryptographic functions) to the workspace and link them together, or a predefined template can be used. The resulting model can be executed, during which the corresponding operations are computed. Some components also have inner visualizations, called presentations.

Plugins in CT2 are programmed with the programming language C#, using the Microsoft .NET framework. For the presentations, WPF user controls are used. A detailed guide can be found on the wiki page for CT2 developers [2].

3.3 Steganography Model

A typical steganography procedure entails hiding the secret data inside a file on the sender's side, sending the file over a communication channel and then extracting this data on the recipient's side.
A basic steganography model is shown in figure 3.1.

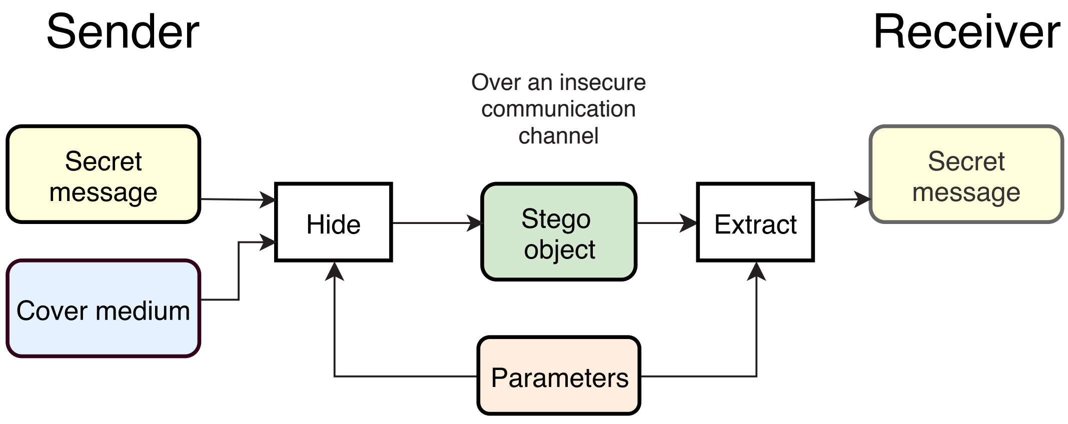


Figure 3.1: Basic steganography model

Components of a basic steganography model [24]:

1. **Secret message:** Data the sender wishes to send secretly, this could be encrypted (ciphertext) or just plaintext.

2. **Cover file:** A non-secret innocent looking text, image, audio or any other type of file in which the message could be embedded in. The original cover file is not necessary to extract the message on the receiver's side. In fact, it is encouraged that the original cover file to be only known to the sender and not available anywhere else. This way the attacker is not able to find it and compare it with the stego object.

3. **Stego object:** The cover file with the secret message embedded in it. This is what the recipient will get over the insecure communication channel.

4. **A hiding function h** where h(m, c, p) = s with:
   m = secret message
   c = cover file
   s = stego object
   p = parameters, depend on the used technique
   This function hides the message in the cover file using some steganography technique.

5. **An extraction function e** where e(s, p) = m
   Input is the stego object the receiver obtains and parameters if any were used to hide the message. Output is the extracted message which could be plaintext or ciphertext; in case it is ciphertext, the extracted message must be decrypted.

In some cases, a steganography key could also be part of the model, the key must be exchanged in order to extract the message successfully. This helps raising the complexity of the hiding process.

3.4 Types of Steganography

Basically, data can be hidden in any type of media. Depending on the type of the cover medium used, steganography can be classified into the following types [21]:

- **Text Steganography:** This type entails hiding text in other text. Approaches used to achieve that are divided into the following categories [3]:
  - Format-based methods
  - Random and statistical generation
  - Linguistic steganography
- **Image Steganography:** Images are an excellent medium to hide data. This lies in the fact that steganography techniques make use of the relatively high amount of redundant bits in an image without causing much visual distortion [5]. More information about this follows in the chapter "Image Steganography".
- **Audio Steganography:** Inaudible frequencies in an audio can be used to conceal data without having effect on how the audio sounds. Alternatively, LSBs of an audio file can also be modified to hide data [21]. Yet it is important to note that audio has a lower hiding capacity than images. This is due to the fact that the

auditory sense is harder to trick than the sense of sight [20, Chapter 7].

- **Video Steganography:** A video is a combination of audio and images. Therefore, videos have the capacity to hide large amounts of payloads [21]. The capacity is the result of combining the hiding capacity of each image (frame) in the video, in addition to the audio hiding capacity when needed.

- **Steganography using computer hard drive:** Another interesting way to hide data is by using the hard drive of a computer [20, Chapter 17]. In a computer's hard drive, many small chunks of memory remain unused due to the fragmentation done by the operating system. Special programs can be used to access these chunks and embed data into them. Moving forward, only the people who know the position can access these files [16]. To achieve high capacity the whole hard drive can be deployed as a steganographic file system using special software and tools [20, Chapter 17].

- **Network Steganography:** Network protocols, e.g., TCP, UDP, IP can be used as cover media in steganography. Moreover, the ISO model with its different layers offers a number of covert channels which can be used to transmit data secretly [21].

In this thesis, image and text steganography were chosen. Reasons include the high availability of images and text on the Internet in form of blogs, text messages, social media posts, etc. Furthermore, many techniques were developed for image and text steganography exploring the interesting ways to modify these to hide data.

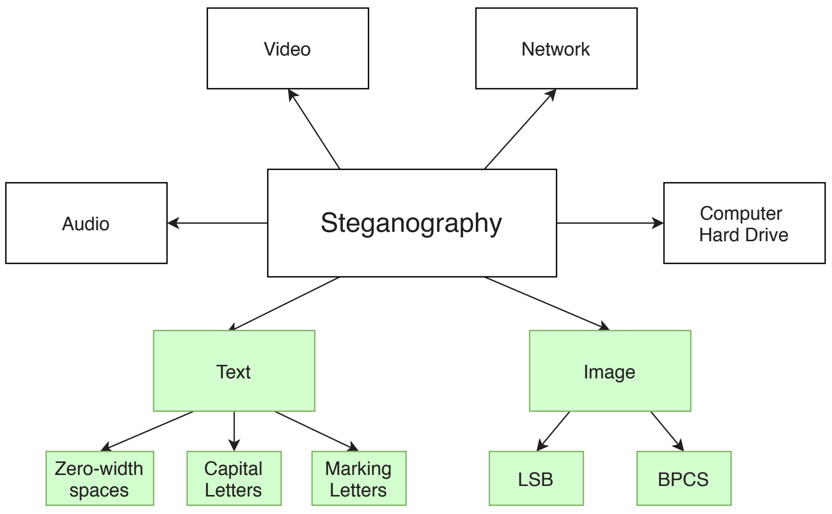


Figure 3.2: Types of steganography

In figure 3.2, the green-colored fields are the focus of this thesis.

Note: There are many more techniques available for image and text steganography. However, the five ones in the bottom row are the techniques implemented within the scope of this thesis.

3.5 Characteristics of Steganographic Techniques

In general terms, there are three main features that can evaluate the quality of a steganographic technique [17]:

- **Hiding capacity:** This represents the maximum size of payload that can be embedded in a cover file. This can be measured relatively to the size of the cover file (percentage), or it can be an absolute value.
- **Perceptual transparency:** This is an essential feature in steganography. When implementing a steganographic method, it is important to make sure that the method does not result in the degradation of the original cover medium. Otherwise third parties who are observing the communication over the channel can figure out that a secret message is being hidden. This defeats the purpose of steganography. Therefore, modifications made to a cover medium should be subtle and not easily to detect.
- **Resistance to steganalysis:** A good steganographic technique should also be able to defend against steganalysis (see chapter 3.6) to some extent. Using techniques that allow customization and the creation of versions only known to the sender and receiver can help against steganalysis that specifically target well-known techniques.

There is a certain trade-off between these characteristics. This is particularly true between hiding capacity and perceptual transparency. In order to achieve very little degradation of the cover medium, only a small amount of data should be embedded (less hiding capacity), and vice versa.

Robustness and tamper resistance can also be considered when evaluating steganographic methods [17], however they are secondary characteristics.

3.6 Steganalysis

While cryptanalysis studies cryptographic techniques to find their weaknesses and to break ciphers, steganalysis studies and exploits the weaknesses of steganographic techniques. Steganalysis starts with a number of files suspected of containing hidden data. The result here is to determine whether any of the files has data encoded in it. In best-case scenarios, the hidden data is retrieved. [7].

Basic approaches for steganalysis rely on statistical analysis. Hereby, the statistics of the examined file and files related to it are analyzed in order to find some inconsistencies or abnormal break of patterns. This step helps obtaining information that is almost impossible to figure out just by looking at the file. The objective here is to compare results and draw some sort of conclusion based on the provided data.

Analyzing files from the same origin, for example pictures taken from the same camera, and/or tracking the original file are excellent ways to obtain a baseline to compare the suspect files to [7].

4 Image Steganography

In this chapter, image steganography and the corresponding plugin in CT2 are discussed. In image steganography, an image is used as a cover file to hide a secret message. Images are considered the most convenient and popular choice to be used as a cover medium due to their high hiding capacity and availability online [5].

4.1 Image Steganography Preliminaries

Digital Images

In the past, image steganography was hiding information by placing symbols, objects, or maybe certain letters in an image without attracting much attention to the viewer. Best practice here was using images with shadows, random patterns, and spots which could realistically take any form or shape [20, Chapter 7].

But in the digital age, the standard is working with bits. To computers, an image is composed of a number of elements called pixels. Each pixel contains information about the color of the image depending on the horizontal and vertical position of the pixel [19].

In grayscale images, a pixel records the intensity of the color ranging between 0 and 255 (8-bit pixel). Black is coded as 0, white as 255 and the rest of the values represent the different shades of gray [26].

In color images, the RGB additive color model is used. The primitive colors red, green, and blue are added together in various amounts to reproduce a broad array of colors [6]. In this color model, a pixel has three channels recording the amount of red, green, and blue in the pixel's position. The value in each channel ranges from 0 to 255. This results in a 24-bit pixel (3 color channels $\times$ 8 bits in each channel).

Image File Formats

Digital images have various formats. Each format has its own characteristics, advantages, and disadvantages. Example of these formats include JPEG, PNG, GIF, BMP. However, the two most used formats are PNG and JPEG. Following is a comparison between these two formats and their compatibility with steganography:

- **Joint Photographic Experts Group (JPEG):** JPEG is a lossy compression format. This means that during compression, redundant information is removed in order to make the file size smaller. This comes in handy when sharing images online to establish faster transmission, or when only small memory capacity is available [12]. But depending on the degree of the compression, an image's original integrity could get lost along with important information for extracting a secret message. This is because steganographic techniques usually make use of these same redundant bits to hide data.
- **Portable Network Graphics (PNG):** PNG is a lossless compression format. With PNG the image retains higher quality and looks a lot sharper. Additionally, no data of the image gets lost with PNG format [12].

As a result, PNG is the preferred choice when using steganography to ensure that all encoded payloads stay preserved in the stego image. Also using JPEG with a low compression degree should work fine as well.

Image Steganography Techniques

Image steganography techniques can be classified into many categories, but the most two prominent are the following:

- **Spatial domain techniques:** The mechanism of these techniques is generally based on substituting bits in the cover image with bits from the secret message in areas of the image where changes are most likely to go unnoticeable. The most commonly used technique here is the LSB method [11]. These techniques are widely used due to their simplicity. On the other hand, one of the disadvantages is their incompatibility with any compression or croping operation performed on the image. This can directly lead to alteration or loss of essential data.
- **Frequency domain techniques:** Also known as transform domain techniques. These techniques start by transforming the cover image and then embedding the data. The transformation is done to help allocate less sensitive areas of an image where the data can be embedded. This helps achieve robustness which is very relevant when working with images with a lossy compression format. These techniques are a bit more complex and require the application of advanced mathematics to some extent [5].

Other categories include statistical techniques, distortion techniques and spread spectrum techniques [11].

4.2 Least Significant Bit

Substituting the least significant bits of pixels' RGB color channels with a secret message's bits is the simplest and most popular technique of image steganography. This is due to the fact that modifying the LSBs of an image's pixels result in very little visual effect on the image [5]. Figure 4.1 is a pixel sample with corresponding RGB values before and after replacing the LSBs with the message bits 111 as an example. Here is can be seen that the color difference between these samples can hardly be detected by the human eye.

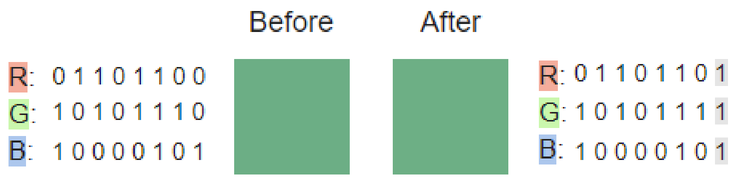


Figure 4.1: Pixel sample before and after LSB modification

In the event that a higher hiding capacity is required, the first k least significant bits can be used to encode a payload. However, this depends on the image because in some cases modifying more than the LSB could lead to image distortion [5]. Consequently, this should be done with caution to avoid the detection of a hidden message.

In the implemented LSB mode in CT2, the user is able to experiment with the selection of bits to substitute and see the changes accordingly. More about this is mentioned in the section "LSB GUI".

In the classical LSB technique: *hiding capacity* = 3 × image width × image height.
In more general terms: *hiding capacity* = number of chosen bits × image width × image height.

4.2.1 LSB Implementation

Following methods were implemented to perform LSB related actions using the programming language C#:

HideDataLSB() This method is called to hide a message using the bitmasks determined by the user. Bitmasks can be chosen using the Choose bits view if the presentation is activated. Otherwise the user can choose the bitmasks in the settings. Provided a valid input cover image, the method starts by converting the input message to a bitarray. Subsequently, it starts looping through the pixels of the cover image, replacing the bits of the pixels with bits of the message given the bit is set to 1 at the same index in the corresponding bitmask. The loop is exited when all the message bits are embedded. The rest of the pixels in the stego image are a copy of the pixels from the original image. In order to extract the message from the stego image correctly, a header with information about the length of the message and bitmasks is created. The last three pixels of the cover image are reserved to encode the header. One pixel is used to store the bitmasks. The other two pixels are used to encode an integer representing the length of the secret message. In figure 4.2 a UML activity diagram is shown representing the process flow of the method.

ExtractDataLSB() When this method is called, the header is extracted from the stego image. The program uses the header information to know the length of the message and which bits were used to hide it. This information is used to extract the message. Lastly, the output component is updated with the result.

4.2.2 LSB GUI

After executing the model—assuming the presentation is activated in the settings—the main menu view is displayed.

From the main menu, the user can switch between different views. These views are intended to help better understand the process and experiment with the selection of bits. These views are shown and explained below using the cover image shown in figure 4.3 and the secret message: "send me your location" as example inputs.

Figure 4.3: Cover image example [18]

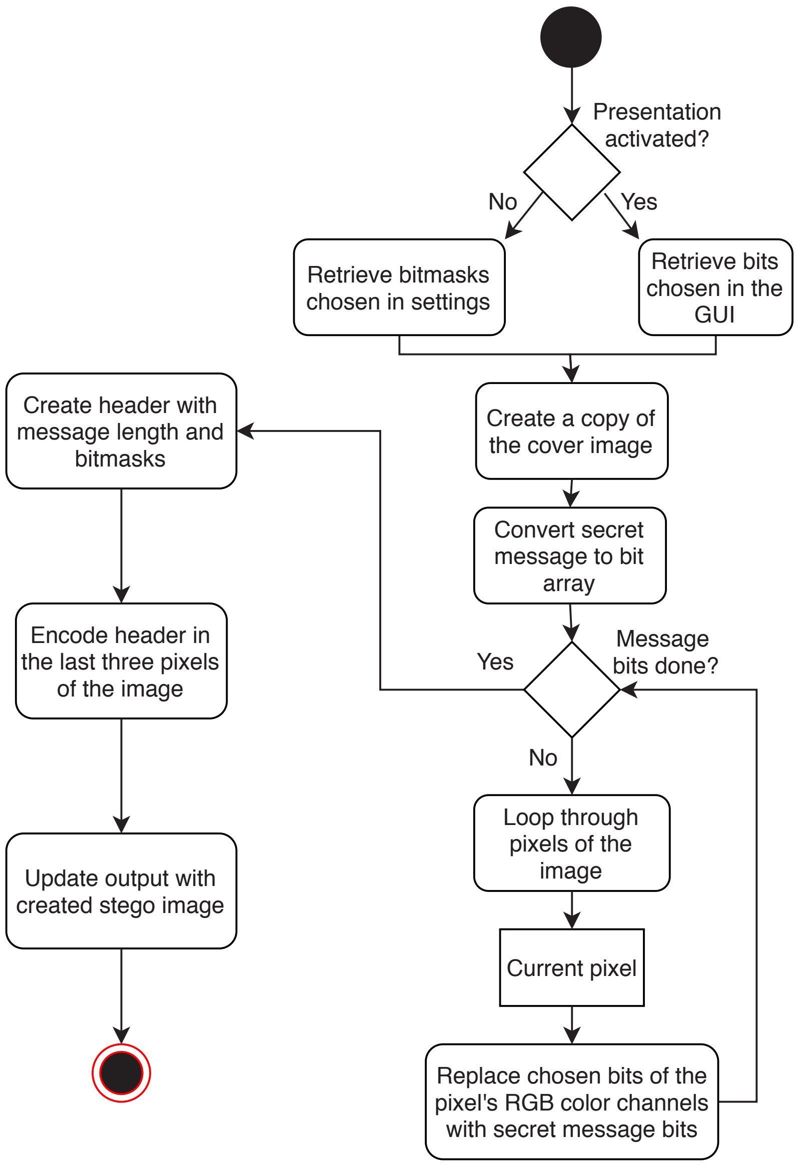

Presentation activated?
No
Yes
Retrieve bitmasks chosen in settings
Retrieve bits chosen in the GUI
Create a copy of the cover image
Convert secret message to bit array
Message bits done?
Yes
No
Create header with message length and bitmasks
Encode header in the last three pixels of the image
Update output with created stego image
Loop through pixels of the image
Current pixel
Replace chosen bits of the pixel's RGB color channels with secret message bits

Figure 4.2: UML activity diagram for the HideDataLSB method[1]

The views are:

[1]Generated with Flowchart Maker & Online Diagram Software

How the algorithm works (Introduction view) This is an introduction view with the goal to give the user some background information about steganography and how it is applied. This view includes four slides which the user can browse through. The first two slides explain what steganography is and what a general model looks like. The other two slides address the concept of the LSB technique.

Choose bits view This view is the experimental part of the visualization. As seen in figure 4.4, there are three groups of buttons. Each group portrays one of the RGB color channels. The displayed buttons (bitmasks) represent which bits are chosen to hide a message (1 = bit is chosen, 0 = bit is not chosen). As default the LSBs are already selected. Buttons of the selected bits have a cyan background color. To select/unselect a bit, the user only has to click on the button of the corresponding bit and it will be enabled/disabled.

In order for the changes to take place, the *Apply Changes* button in the below bar should be clicked. This action will execute the *HideDataLSB()* method. The output stego image and other panels get updated accordingly. With this view the user can keep adding bits

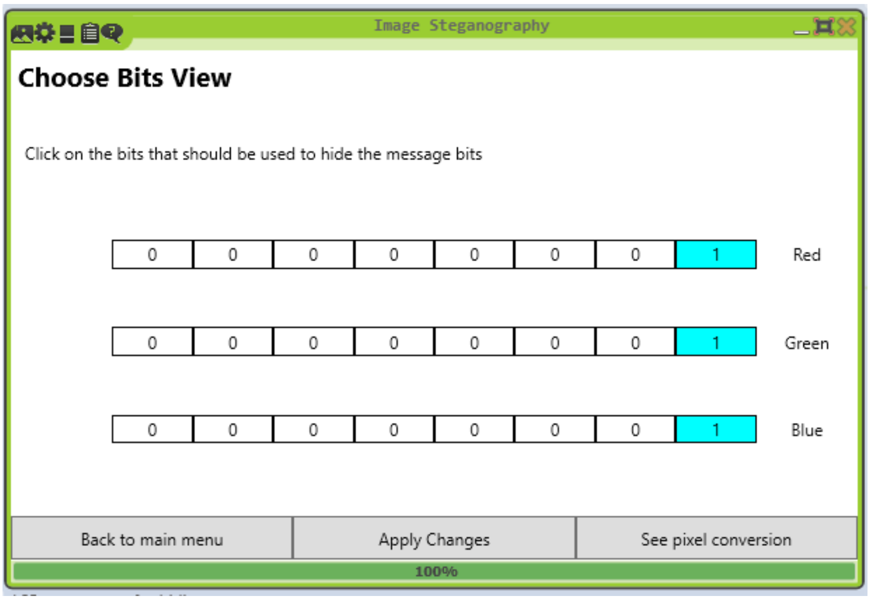


Figure 4.4: Choose bits view

to increase the hiding capacity and observe the results. As long as there is no noticeable image distortion, more bits can be added or removed as needed.

Note: If the presentation is activated, no hiding process will be done until the bits are chosen in this view and the *Apply Changes* button is clicked.

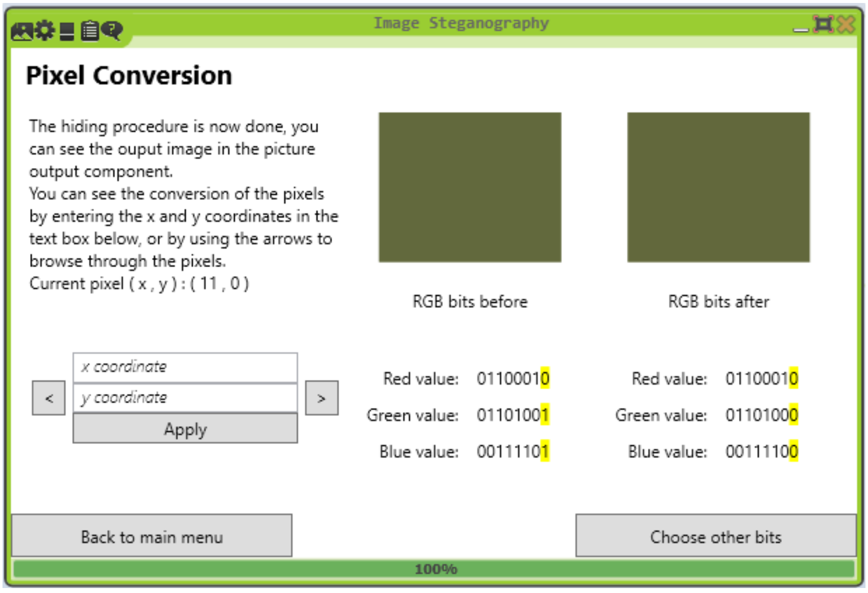
Pixel conversion view After the hiding process is executed, the button for the pixel conversion view is activated. This view serves as a way to examine the difference between the original and the stego image by viewing color samples of each pixel before and after the substitution. Additionally, the bits of the RGB values are displayed while highlighting the bits that were chosen for the hiding process. As seen in figure 4.5, the pixel currently shown have the coordinates: (11 , 0).

Displayed on the left side is a color sample of the pixel before the conversion, on the right a color sample of the pixel afterwards. In this example only the LSBs were chosen, therefore there is no noticeable change between the colors.

The user can browse through pixels by using the next and previous arrows. Alternatively, the user can enter the coordinates manually to jump to a certain pixel. This can be done by entering the x and y coordinates in the corresponding text boxes and then clicking the *Apply* button. If the values are valid, the view is updated accordingly. A valid input means entering a value in the following ranges:

- 0 <= x coordinate < image width
- 0 <= y coordinate < image height

If these criteria are not met, a warning appears that the input is not valid and the valid ranges are shown to the user.



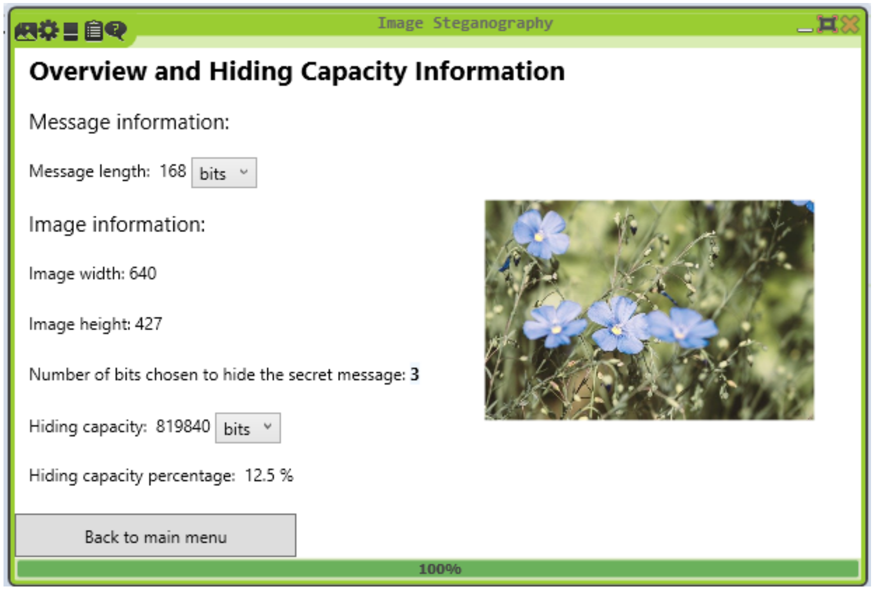
The hiding procedure is now done, you can see the ouput image in the picture output component.
You can see the conversion of the pixels by entering the x and y coordinates in the text box below, or by using the arrows to browse through the pixels.
Current pixel (x , y) : (11 , 0)

	RGB bits before	RGB bits after
Red value:	01100010	01100010
Green value:	01101001	01101000
Blue value:	00111101	00111100

Figure 4.5: Pixel conversion view

Overview and hiding capacity information This view helps the user get a general overview of the inputs provided. This includes: the secret message, the cover image, chosen bits and hiding capacity as shown in figure 4.6.

Firstly, the size of the secret message in bits is denoted. Using the adjacent combobox, the size of the message can be converted to one of the following units: *bits, bytes, kb, mb.* Next comes the image height, width, and number of chosen bits. The hiding capacity is also computed and displayed, this value depends on the number of chosen bits and the size of the cover image. This value can also be converted to different units using the provided combobox. On the right side is the original cover image as a reference.



Overview and Hiding Capacity Information

Message information:

Message length: 168 bits

Image information:

Image width: 640

Image height: 427

Number of bits chosen to hide the secret message: 3

Hiding capacity: 819840 bits

Hiding capacity percentage: 12.5 %

Figure 4.6: Overview and hiding capacity information

After stopping the program, the buttons in the main menu view are disabled except for the introduction view, as it is a general view and not dependent on the input image. In order to re-enable the buttons, the program should be started again.

4.3 Bit Plane Complexity Segmentation

Bit Plane Complexity Segmentation (BPCS) is a steganography technique introduced in a paper written by Kawaguchi & Eason to overcome the shortcomings of other steganography techniques with regards to the hiding capacity. This technique can hide up to 50% of the data amount in the cover image [13]. The concept underlying this technique is based on the characteristics of the human vision, where substituting complex regions in an image with other complex regions does not affect how the image is perceived.

Bit slicing a multi-valued image results in a set of n bit planes ($\mathrm{p}_1, \ldots, \mathrm{p}_\mathrm{n}$), where p_1 is the plane with most significant bits of an image and p_n with the least significant bits of the same image. With this definition, the complexity of the planes increases from MSB to LSB. In the paper [13], the authors discuss how to divide the regions in an image and how to classify the regions into informative or noise-like regions.

A multi-valued image consisting of 24-bit pixels can be decomposed into a set of 24 of bit planes. Whereas a grayscale image can be decomposed into a set of 8 bit planes. Each one of these planes is then divided into local areas, the authors found that 8×8 is a good size for these local areas (blocks).

Block Complexity Although there are several methods available to measure the complexity of a binary image block, border length of an image was chosen for this technique. With this approach the complexity of a binary image is computed as following [13]:

$$\alpha = \frac{\text{number of B-W changes in the image}}{\text{max of all possible B-W changes in the image}}$$

Therefore, the result α should be in the range [0,1] with 0 being not complex at all and 1 being totally complex (checkerboard pattern).
Note: B-W changes means the changes between 0 and 1 in the rows and the columns of the binary image.

Figure 4.7 displays examples of two image blocks. On the left side is an image block with a complexity of 0. The block on the right has a complexity of 1. These are the extreme cases where the image is totally simple in the first case and totally complex in the second one. In real life the patterns range widely. For 8×8 sized blocks there are 2^{64} possible patterns. A histogram was made to visualize the distribution of the complexity of these binary patterns. The result was a normal distribution with an average of 0.5 which can be seen in Fig.2 of the original paper [13].

After computing the complexity, an image block can be classified either as an informative region or noise-like region. This is done by comparing the block's complexity with some complexity threshold α_0, this value could be chosen freely but a typical value is 0.3. The complexity threshold can be modified in the settings of the CT2 plugin in order to see

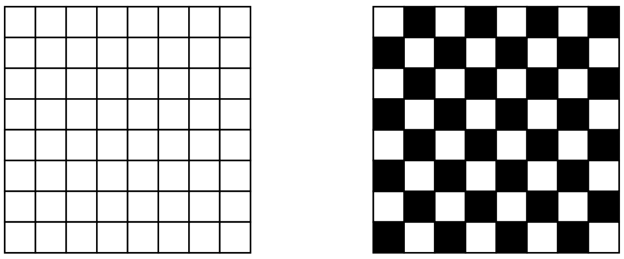


Figure 4.7: Simple vs. complex image blocks

different effects using different thresholds. Increasing the threshold would result in a lower hiding capacity but less image distortion, whereas lowering the threshold can lead to some image distortion but a higher hiding capacity.

Conjugation Binary blocks representing the secret message that will replace the complex blocks in an image are called message blocks. These message blocks should be complex as well. If that is not the case, a message block should undergo a conjugation operation in order to sustain the complexity level of the image block it replaces.

An example of the conjugation operation can be seen in figure 4.8. In this example, P is the original message block, P* is the conjugated version and W_c is the checkerboard pattern that helps increase the complexity of the message block.

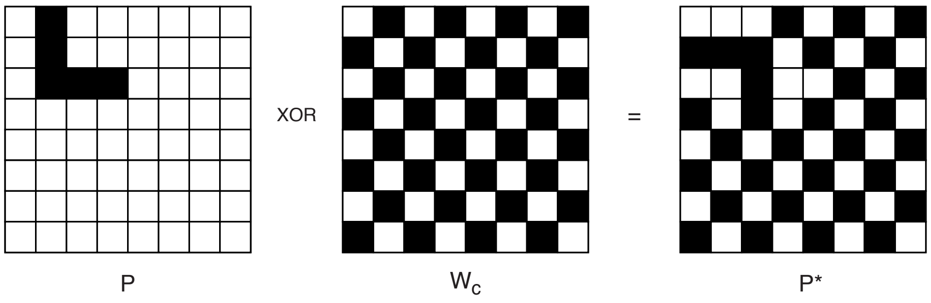


Figure 4.8: Conjugation operation

Following rules apply for the conjugation operation [10]:

- $P^* = P \oplus W_c$

- (P*)* = P
- α(P*) = 1 - α(P)
  With α(P) as the function that computes the complexity of a block P.

Note: the last rule applies theoretically. However, in the implemented plugin in CT2, this might not always apply because the first bit is used to indicate if the block is conjugated or not. More about that in the section "BPCS Implementation".

Canonical Gray Coding There are two binary coding systems: Pure Binary Coding (PBC) and Canonical Gray Coding (CGC). An image originally uses PBC for the coding of its pixels. However in the BPCS technique, CGC is preferred over PBC, as it achieves a "better looking" (blocking-less) stego image [14]. This means when replacing the complex image blocks with message blocks, the changes in the bits lead to less noticeable color changes when using CGC. Therefore, CGC leads to less image distortion which is an important aspect in steganography. On the other hand, using CGC might result in a lower hiding capacity [13].

PBC and CGC are related by an $\oplus$ operation in the following way [14]:
$g_1 = b_1$
$g_i = b_{i-1} \oplus b_i$
g_i is the i-th bit of a pixel byte in CGC and b_i is the i-th bit of a pixel byte in PBC.

The authors of the paper also pointed out the fact that a program that implements BPCS can be customized and parameters can be modified to create a custom program known only to the sender and receiver. By setting and agreeing on some parameters, the sender and receiver can reduce the chance of attackers using standard settings to extract the secret message. The ability to modify the complexity threshold and the sequence of the color layers is offered in the CT2 component's settings.

4.3.1 BPCS Implementation

For the implementation of the BPCS algorithm, several classes were introduced to manage the different bit planes and perform the hiding process.

An overview of these classes with their important attributes and methods can be seen in figure 4.9. Following is a more detailed description of each class:

Pixel This class stores the RGB information of the cover image's pixels. One of the attributes is a bool array that stores the 24 CGC bits of a pixel. The CGC bits are used later to construct the 24 different image bit planes. The order of these bits in the array depends on the color order selected by the user in the settings. After the complex blocks

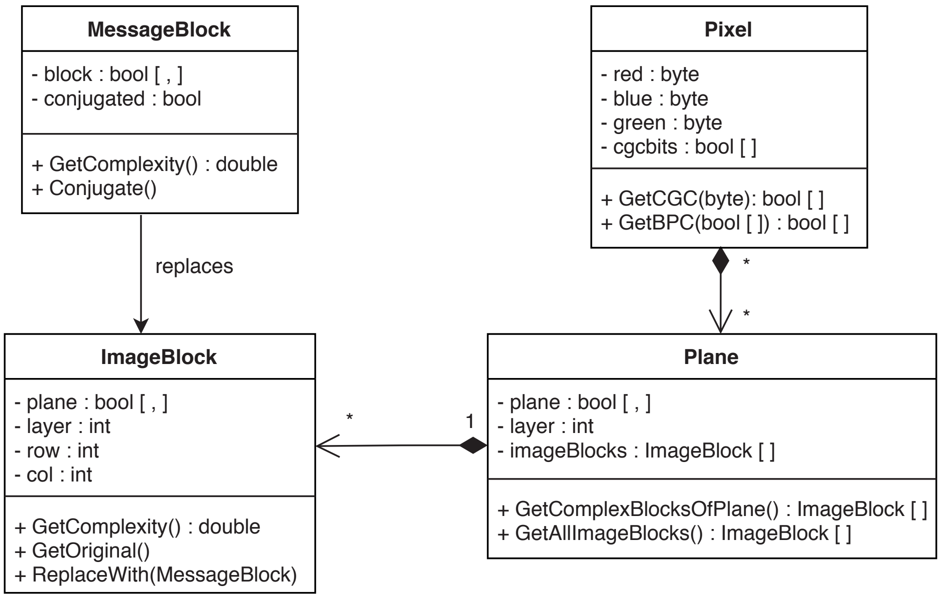


Figure 4.9: UML diagram of classes implemented for BPCS algorithmd[2]

are replaced with the message, the original PBC format can be retrieved from this class to build the stego image. An instance of this class requires the Color instance from the image's pixels and the order of the colors as parameters.

Plane This class represents the different bit planes of the cover image. Attributes include the layer of the plane and a two-dimensional bool array that represents the bits of the plane. The method *GetComplexBlocksOfPlane()* computes the complexity of each block and returns an array with the complex blocks. The method *GetAllImageBlocks()* returns an array with all blocks of the plane.

ImageBlock This class is used to store information about the image blocks. Defining a bit plane, layer, column and a row results in a specific image block. Complexity can be computed for each image block to determine later whether the image block is complex or not.

[2]Generated with Flowchart Maker & Online Diagram Software

MessageBlock This class is used to create the blocks that represent the input secret message. These blocks are used later to replace the complex image blocks. Attributes here are a two-dimensional bool array that stores the bits of the message and a bool to indicate whether the block is conjugated or not (this is necessary for the extraction process). The constructor has two versions: One is for the header message block containing the length of the secret message. This version uses all 64 bool values to store a 64-bit integer representing the length of the message. The second version is to store the actual bits of the message. This version only stores 63 bit values, this is because the first bit is reserved to indicate whether the message block is conjugated or not. After computing the complexity, if the block is complex, the first bit is set to 0. Otherwise, the block is conjugated and the first is set to 1.

Following methods are implemented for the hiding/extracting procedure:

HideDataBPCS() This method starts by dividing the message bits into 63 bool arrays and creates a message block for each array. These message blocks along with the header message block are added to an array. Secondly, the 24 bit planes are defined by using the CGC bits of the pixels. The order of the planes depends on the configuration chosen in the settings. Complex blocks of the planes are retrieved using the *GetComplexBlocksOfPlane()* method. These complex blocks are then replaced by the message blocks. To hide the complete message, the number of complex blocks should be greater than or equal to the number of message blocks. If this is not the case, a warning will be displayed informing the user that the capacity was not enough to hide the entire message. Possible solutions here: lowering the complexity threshold or experimenting with other images with more complex regions, see section 4.3.3 for more information on how to increase the hiding capacity. Lastly, the stego image is built by fetching the PBC bits of the pixels and the output component is updated with the new stego image. A visual demonstration of this method can be seen in figure 4.10.

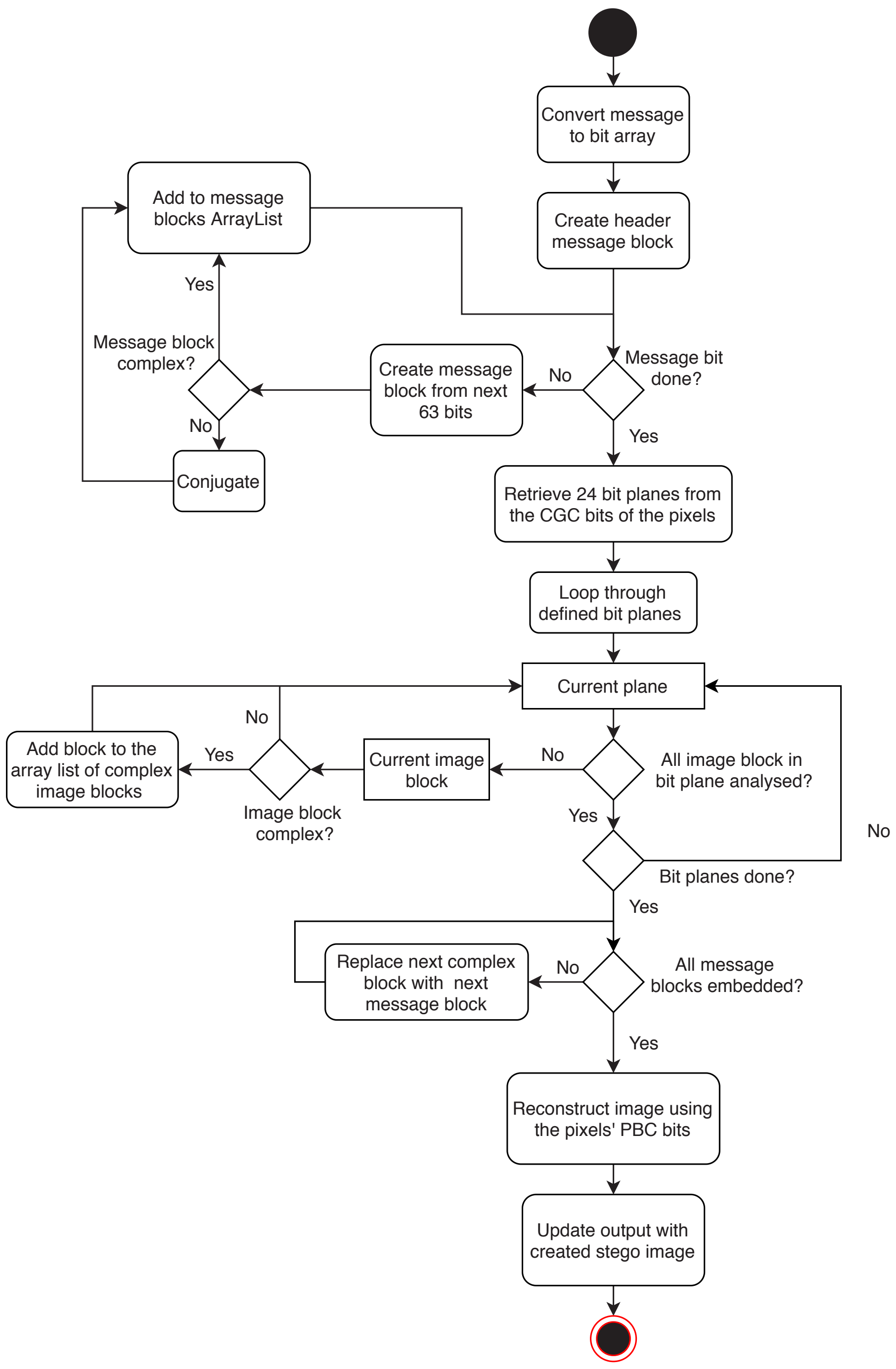

Convert message to bit array
Create header message block
Add to message blocks ArrayList
Yes
Message block complex?
No
Conjugate
Create message block from next 63 bits
No
Message bit done?
Yes
Retrieve 24 bit planes from the CGC bits of the pixels
Loop through defined bit planes
Current plane
No
Add block to the array list of complex image blocks
Yes
Image block complex?
Current image block
No
All image block in bit plane analysed?
Yes
No
Bit planes done?
Yes
Replace next complex block with next message block
No
All message blocks embedded?
Yes
Reconstruct image using the pixels' PBC bits
Update output with created stego image

Figure 4.10: UML activity diagram for the HideDataBPCS method [3]

ExtractDataBPCS() In order for this method to extract the secret message correctly, the same complexity threshold and color layer order should be used in the hiding/extraction component. Similarly to the *HideDataBPCS()* method, planes and complex regions of the stego image are retrieved. After that, the message bits are collected by knowing the length of the message using the first block (header message block). If the message block is conjugated (first bit set to 1) the *GetOriginal()* method is called. Lastly the bits are converted to a string and the output secret message component is updated with the result.

4.3.2 BPCS GUI

In this section, the presentation for BPCS is presented along with the different views. Provided that the presentation is activated in the settings, the main menu view is shown after executing the model. The main menu contains buttons to navigate between these different views:

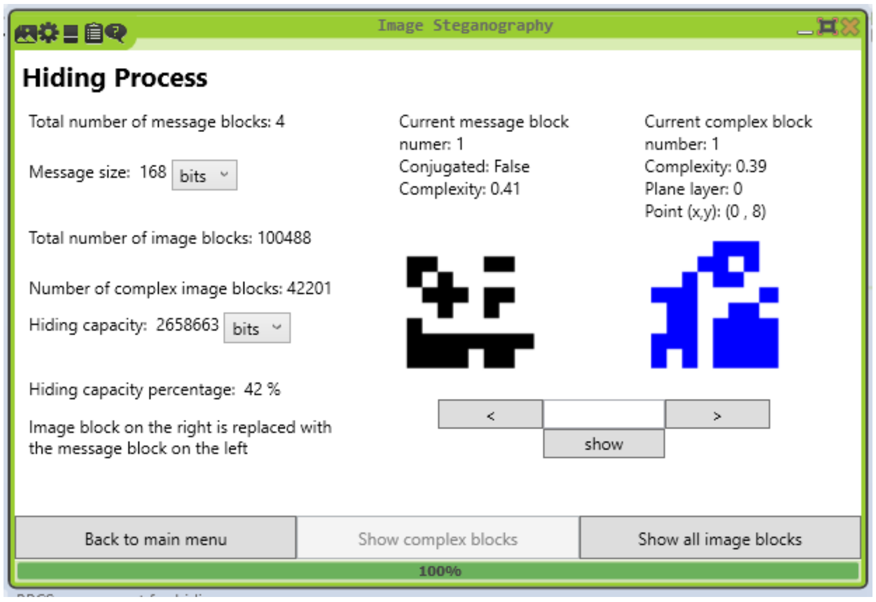
How the algorithm works (Introduction view) Just like in the LSB mode, this view gives the user an introduction to steganography and the concept behind the BPCS approach. An example of a complex and an informative region is shown, in addition to an illustration of the conjugation operation.

Hiding process view This view offers the user a concrete visualization of how the image blocks and message blocks look like, see figure 4.11.

On the left side, some general information about the message size and hiding capacity depending on the chosen configurations in the settings is displayed with the ability to convert the sizes into different units using the combobox. On the right side, two columns are located to display complex image blocks (hider blocks) and the message blocks they were replaced with. The user can iterate through the complex blocks either by using the arrows provided below or by entering a valid value in the textfield and then clicking the *show* button. A valid value here means a number between 0 and number of complex image blocks shown on the left side. If an image block is not replaced by a message block at the current iteration, a *no message block* label will appear.

There is also the option to see all image blocks (not only complex blocks) by clicking the button in the below bar *Show all image blocks* that navigates to the second panel as seen in figure 4.12. The user can browse through these blocks in the same way described above.

[3] Generated with Flowchart Maker & Online Diagram Software



Total number of message blocks: 4

Message size: 168 bits

Total number of image blocks: 100488

Number of complex image blocks: 42201

Hiding capacity: 2658663 bits

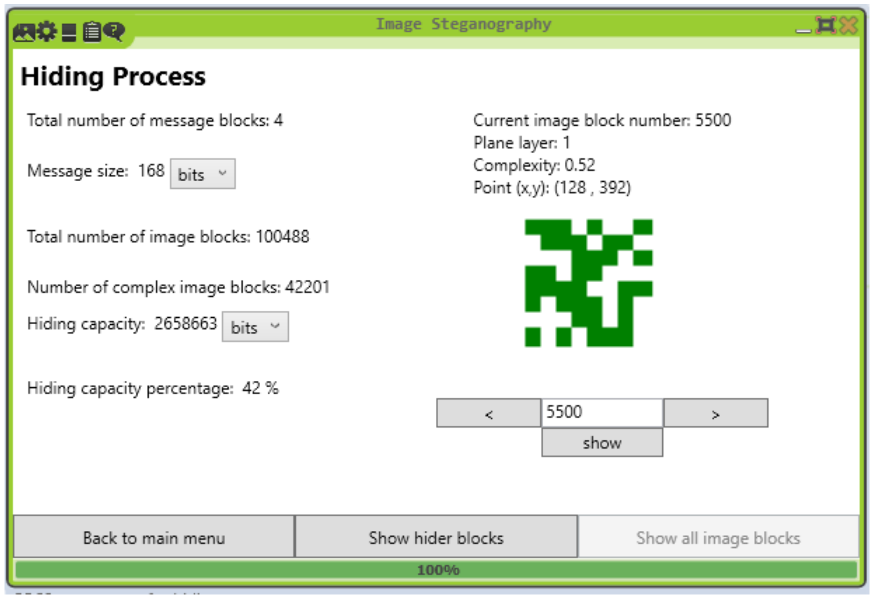
Hiding capacity percentage: 42 %

Image block on the right is replaced with the message block on the left

Current message block numer: 1
Conjugated: False
Complexity: 0.41

Current complex block number: 1
Complexity: 0.39
Plane layer: 0
Point (x,y): (0 , 8)

Figure 4.11: Hiding process view, complex image blocks only



Total number of message blocks: 4

Message size: 168 bits

Total number of image blocks: 100488

Number of complex image blocks: 42201

Hiding capacity: 2658663 bits

Hiding capacity percentage: 42 %

Current image block number: 5500
Plane layer: 1
Complexity: 0.52
Point (x,y): (128 , 392)

Figure 4.12: Hiding process view, all image blocks

Pixel conversion view This view serves the same purpose as the Pixel conversion view in LSB mode.

Bit planes view: In order to better visualize and understand the difference between informative and noise-like regions, this view displays the 24 different bit planes of the cover image. Each 8 planes with the same color channel (red, green or blue) are grouped together and shown from MSB to the LSB. Using the below bar, the different groups can be seen and compared. Figure 4.13 shows the blue bit planes of the cover image example 4.3.

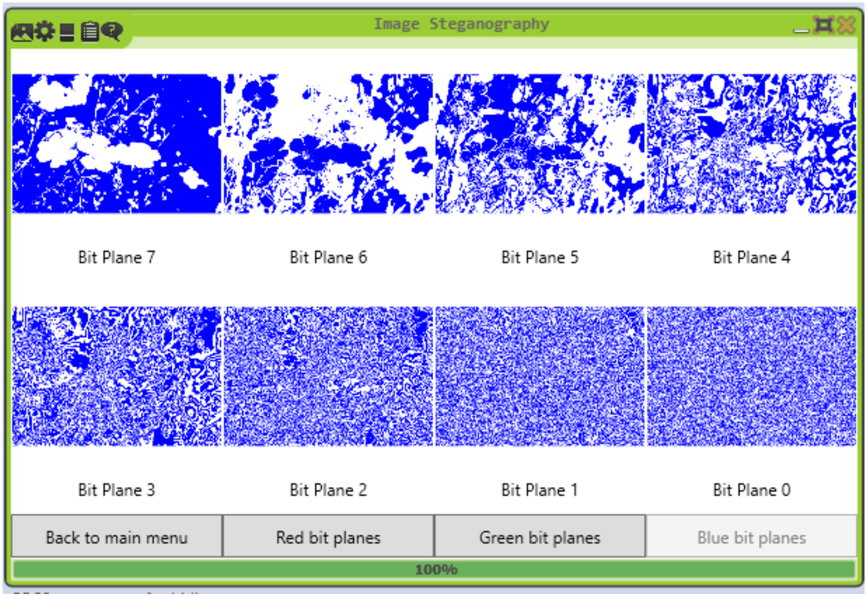


Figure 4.13: Bit planes view

4.3.3 Some Results Based on Testing Different Input Images

The authors of the paper that introduced the BPCS algorithm also shared some results they obtained by using the experimental system they developed. One of these results states that increasing the sharpness of the cover image can lead to a higher embedding capacity [13]. I tested this statement in CT2 using three versions of the same image: The original image, a slightly blurred version, and a slightly sharpened version of the image. The outcomes, which can be found in the table 4.1, confirm this predicate.

Also during testing, another factor seemed to play a significant role in determining the hiding capacity of an image. This factor was how detailed an image was. The more detailed and the more colors, items or objects that were displayed in an image, the higher embedding capacity it had. This is due to the fact that detailed images contain much more complex image blocks than a simple image with mostly solid background colors. Therefore, when a high embedding capacity is required, it is suggested to stay away from images of landscapes, skies, or images with large areas of solid colors in general. This difference can be demonstrated when comparing the results of the three test images. Test image 1 [18] is an image of a landspace with big areas of solid colors, Test image 2 [18] has some detail in it, and Test image 3 [18] is an image of a street with a lot of detail in the buildigs, cars, etc. All these images have the same dimensions and the results are based on tests using the same complexity threshold for all images.

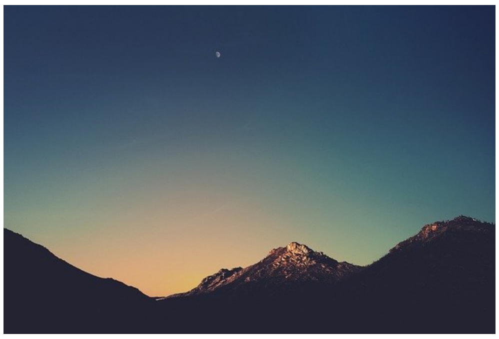


Figure 4.14: Test image 1 [18]

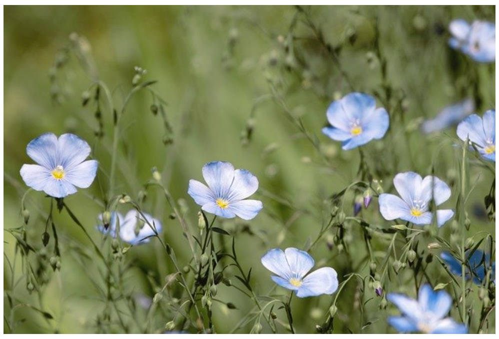


Figure 4.15: Test image 2 [18]

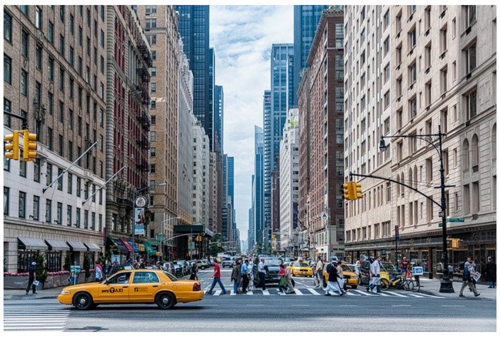


Figure 4.16: Test image 3 [18]

	Example Image 1	**Example Image 2**	**Example Image 3**
Blurred	5.73%	29.72%	65.56%
Original	7.56%	37.11%	73.36%
Sharpened	10.78%	41.84%	75.72%

Table 4.1: Hiding capacity relative to image size (percentage)

5 Text Steganography

Text steganography uses non-secret plaintext as a cover medium. Text can be preferred when choosing a cover medium as it requires less memory than other files, easy to communicate and highly available online in form of blogs, text messages, etc [3]. On the other hand, it is considered one of the trickiest alternatives due to the lack of redundant information which is present in images, videos or audio [15]. In text, even the smallest changes like changing the format of some letters or adding special characters can be directly detected by the reader [3]. Therefore, text steganography should be used with caution, especially when hiding sensitive information, but with the right techniques it can be a practical choice.

5.1 Zero-Width Spaces

One of the most popular techniques for text steganography is white space steganography. This is because white spaces cannot be directly detected and it does not effect any characters of the text. However, too many visible white spaces in a text can raise some suspicion, hence using zero-width spaces as presented in this section. In UTF8 encoding, there are a number of zero-width space characters. These are non-printing characters, intended to indicate word boundaries or line breaks in scripts that do not use explicit spaces or in some languages [8].

In the zero-width spaces mode implemented in CT2, the characters '\u200B' and '\u200C' are used for the hiding process. In the settings, the user can enter an offset that indicates where these zero-width spaces should be added. This value should be the same when hiding and when extracting the message. By default this is set to 0 which means the spaces are added at the beginning of the cover text. The offset can be any number between 0 and the length of the cover text, otherwise a warning is displayed. This process starts with converting the secret message to a bit array. Looping through this array, the character '\u200B' is appended when there is a 1 at the same index, or the character '\u200C' when the bit is 0. These characters are added to the cover text starting from the offset set by the user. After the loop is done, the rest of cover text is appended. The result stego text appears exactly like the cover text. But even though these zero-width spaces are not visible, the number of characters in the stego text equals the number of the characters in the cover text plus the number of the bits of the secret message. In the presentation of the component, the cover text is shown along with the placement of the spaces.

Advantages:

– No restrictions regarding the length of the secret message or its characters.

– Any text could be used as cover text.

– No changes can be detected by someone reading the text.

Limitations:

– Can be achieved only when using UTF8 encoding.

– Added spaces can get lost during transmission of the text.

An example would be:
Secret message: hello (40 bits)
Cover text: the weather is nice today (25 characters)
Stego text: the weather is nice today (65 characters)

Figure 5.1 shows the mentioned example executed in CT2 with offset set to 0. The green and yellow blocks represent the spaces added which are not visible in the stego text. The user can also choose to see the bits of the secret message to better visualize how the spaces were added. This can be seen in figure 5.2.

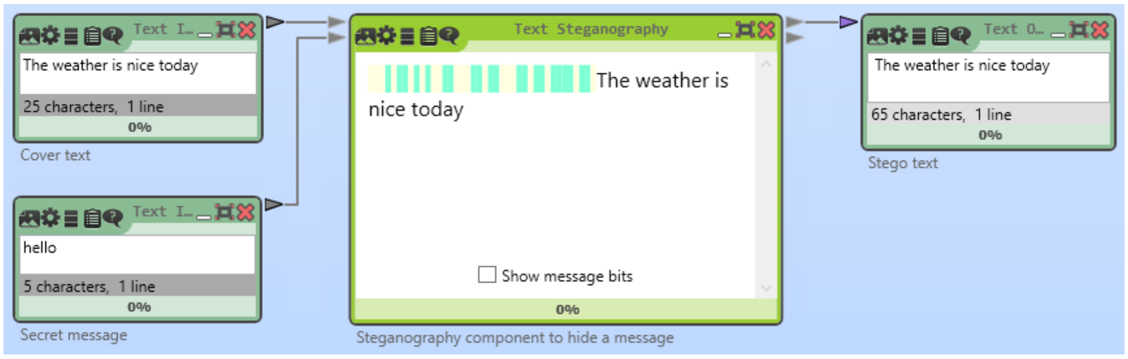
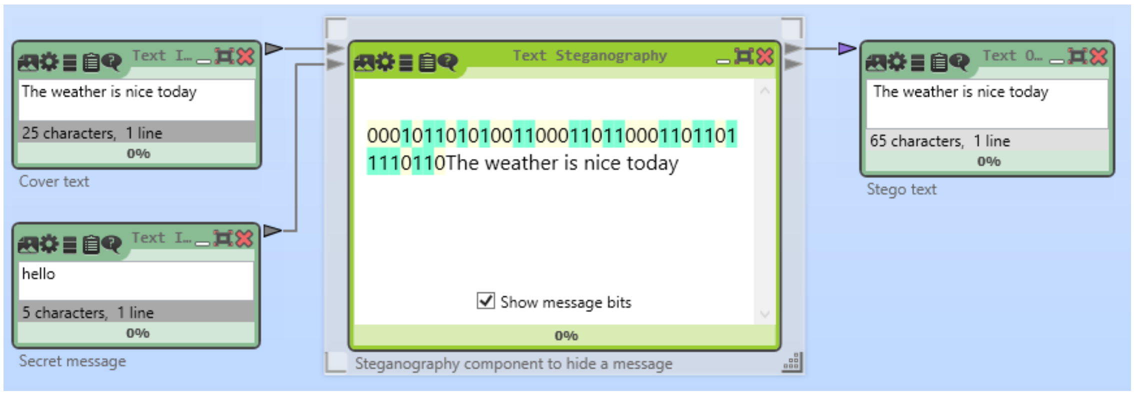


Figure 5.1: Zero-width spaces mode example

5.2 Capital Letters

Using capital letters in a cover text is quite easy to detect, nevertheless it is a very simple technique and can be convenient in some cases. This section discusses the two variants



The weather is nice today
25 characters, 1 line
0%

hello
5 characters, 1 line
0%

000101101010011000110110001101101
1110110The weather is nice today

☑ Show message bits
0%

The weather is nice today
65 characters, 1 line
0%

Figure 5.2: Zero-width spaces mode example with message bits displayed

that have been implemented in CT2 using this technique:

- **Capital letters text:** This mode starts by converting the secret message and cover text to lowercase and then converts the letters of the secret message in the cover text to uppercase. This is a very simple technique but it is very easy to realize the message just by looking at the text. In order to reduce the risk of this happening, it is advised to use a long cover text to distribute the letters of the secret message and make it harder to detect it. In order to hide the whole message correctly, the cover text should include all the letters of the message distributed and in the same order. The secret message should only contain letters and spaces, other characters like punctuation marks and numbers will be ignored.

  Example 1
  Secret message: hey
  Cover text: i am going to the mall today
  Stego text: i am going to tHE mall todaY

- **Capital letters binary:** This is a more discrete variant of using capital letters to encode a message. With this technique, uppercase and lowercase letters in a cover text are used to encode bits. After converting the cover text and secret message to lowercase, the message is converted to a bit array. Looping through this bit array, if the bit is 1 at the current index, the letter at the same index in the cover text is converted to uppercase, otherwise the letter stays lowercase. For a person reading the resulting stego text, it is fairly hard to figure out the hidden text, as the use of uppercase seems quite random since the bits are being encoded and not the letters of the message itself.

With this mode, any character can be hidden. The hiding capacity depends only on the number of letters in the cover text, numbers and other characters are not used. Therefore, to hide all of the message correctly, the number of letters in the cover text should be equal to or greater than the number of bits in the secret message.

Example 2
Secret message: hi
Bits to hide: 00010110 10010110 (16 bits)
Cover text: the dog is in the kitchen
Stego text: the DoG Is In tHe KItchen

5.3 Marking Letters

Marking the letters in a cover text is another alternative for hiding a secret message which could be achieved by using special characters available in UTF8 encoding. Some of these characters include dots added over or under the letters from other languages like Arabic and Vietnamese.

In CT2, two modes are available using this technique. These modes implement the same concepts mentioned in the Capital Letters section, they also have the same advantages and limitations.

- **Marking letters text:** Letters of the secret message are marked in the cover text. Using the same inputs provided in Example 1, the output stego text would look like this:
  i am going to tḥẹ mall todaỵ
- **Marking letters binary:** Letters of the cover text are marked when there is a 1 at the same index in the bit array of the secret message. For Example 2 the output stego text would be:
  the ḍog̣ ịs ịn tḥe ḳịtchen

When choosing one of these two modes, the user can also choose whether the dot goes under or over the letters in the stego text. The same option should be selected when extracting the message.

6 Conclusion

This chapter provides a summary of this thesis while reflecting on the objectives mentioned in the first chapter and how they were achieved.

6.1 Summary

In the introduction, the term steganography was introduced in addition to a motivation section which explains why steganography is important in some situations and how it can be used to add an additional layer of security.

To better understand steganography and what was implemented in CT2, Chapter 3 (Fundamentals) provided clarification about basic concepts regarding steganography. This included the steps needed to use steganography which was demonstrated in the steganography model. After that came the various types of steganography and how it is classified based on the type of medium used as a cover. This also included why image and text steganography were chosen as the main focus of this thesis and the new components in CT2. Characteristics of steganographic techniques were also presented and how they can assist in evaluating a technique. Lastly, steganalysis was briefly introduced in addition to its basic approaches to exploit steganographic techniques.

Chapter 4 (Image Steganography) started by explaining what a digital image is and compared the compatibility of different image file formats with steganography. This was intended to achieve a better comprehension of how the LSB and BPCS algorithms work. Then two sections followed dedicated to discuss the two techniques implemented in CT2: LSB and BPCS. Detailed explanation of the implementation of each technique was provided along with several diagrams that illustrate the code. The GUI sections presented the visual presentation of these components in CT2 and explained the purpose of each view. Screenshots of these GUIs were also added to directly see an example of each mentioned view. At the end some results of testing several input images were shared to point out which factors in an image played a role in increasing the hiding capacity when using the BPCS technique.

Chapter 5 dealt with text steganography and its various modes that were implemented in CT2. Zero-width spaces, capital letters, and marking letters were each introduced individually coupled with the advantages, limitations and conditions for each mode.

6.2 Future Work

Even though this thesis achieved the objectives that were planned, there are still a lot more topics in the field of steganography that can be researched and even implemented in CT2 in the future. These include implementing other types of steganography, for example video steganography which combines approaches applied in image and audio steganography. There are several aspects here that can be visualized to help people who are interested in steganography to better comprehend how the secret messages get hidden. Additionally, diving into the topic of steganalysis and testing the accuracy of its different approaches could make an excellent topic for a master's thesis as this is a more complex topic and requires more time to complete than available for a bachelor's thesis.

Bibliography

[1] CrypTool 2. *CrypTool 2 - CrypTool Portal*. URL: https://www.cryptool.org/en/cryptool2 (visited on 09/10/2020).

[2] CrypTool 2. *Wiki for CrypTool 2 developers*. URL: https://www.cryptool.org/trac/CrypTool2/wiki/WikiStart (visited on 09/10/2020).

[3] Monika Agarwal. "TEXT STEGANOGRAPHIC APPROACHES: A COMPARISON." In: Department of Computer Science and Engineering, PDPM-IIITDM, Jabalpur, India. 2013.

[4] Alan Siper, Roger Farley, and Craig Lombardo. "The Rise of Steganography." In: Pace University. 2005.

[5] Arpit Bhayani. *Everything that you need to know about Image Steganography*. 2020. URL: https://www.codementor.io/@arpitbhayani/internals-of-image-steganography-12qsxcxjsh (visited on 09/10/2020).

[6] Wikipedia contributors. *RGB color model*. 2020. URL: https://en.wikipedia.org/w/index.php?title=RGB_color_model&oldid=974705089 (visited on 09/01/2020).

[7] Wikipedia contributors. *Steganalysis*. URL: https://en.wikipedia.org/wiki/Steganalysis (visited on 09/05/2020).

[8] Wikipedia contributors. *Zero-width space*. URL: https://en.wikipedia.org/wiki/Zero-width_space (visited on 07/30/2020).

[9] *Examining The Importance Of Steganography Information Technology*. URL: https://www.ukessays.com/essays/information-technology/examining-the-importance-of-steganography-information-technology-essay.php?vref=1 (visited on 09/05/2020).

[10] Gediminas19. URL: https://github.com/Gediminas19/BPCS-Steg-Java (visited on 07/20/2020).

[11] Nagham Hamid et al. "Image Steganography Techniques: An Overview." In: *International Journal of Computer Science and Security* 6 (June 2012), pp. 168–187.

[12] *JPG vs. PNG: Which Should I Use?* 2020. URL: https://www.techsmith.com/blog/jpg-vs-png/ (visited on 08/15/2020).

[13] Eiji Kawaguchi and Richard O. Eason. "Principles and applications of BPCS steganography." In: *Multimedia Systems and Applications*. Ed. by Andrew G. Tescher et al. International Society for Optics and Photonics. SPIE, 1999, pp. 464 –473. DOI: 10.1117/12.337436. URL: https://doi.org/10.1117/12.337436 (visited on 08/22/2020).

[14] Eiji Kawaguchi and Richard O. Eason. *Two Binary Number Coding Systems*. URL: http://datahide.org/BPCSe/pbc-vs-cgc-e.html (visited on 07/25/2020).

[15] R. B. Krishnan, P. K. Thandra, and M. S. Baba. "An overview of text steganography." In: *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*. 2017, pp. 1–6.

[16] Paul Marks. *Covert hard drive fragmentation embeds a spy's secrets*. 2011. URL: https://www.newscientist.com/article/mg21028095-200-covert-hard-drive-fragmentation-embeds-a-spys-secrets/#:~:text=Thereisnowaway,160-gigabyteportableharddrive. (visited on 09/01/2020).

[17] Olguin. *Steganography... what is that?* 2016. URL: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/steganography-what-is-that/ (visited on 08/25/2020).

[18] *pixabay: Stunning free images & royalty free stock*. URL: https://pixabay.com/.

[19] Kelvin Salton do Prado. *Steganography: Hiding an image inside another*. URL: https://towardsdatascience.com/steganography-hiding-an-image-inside-another-77ca66b2acb1 (visited on 07/25/2020).

[20] Klaus Schmeh. *Versteckte Botschaften: Die faszinierende Geschichte der Steganografie*. Hannover: Heise / Telepolis, 2009.

[21] Harjit Singh. "Analysis of Different Types of Steganography." In: Department of Computer Science, Punjabi University. 2016.

[22] *spammimic*. URL: https://www.spammimic.com/credits.shtml (visited on 06/30/2020).

[23] *Steganography*. URL: https://en.wikipedia.org/wiki/Steganography (visited on 09/10/2020).

[24] *Steganography Tutorial: A Complete Guide For Beginners*. 2020. URL: https://www.edureka.co/blog/steganography-tutorial (visited on 08/10/2020).

[25] Guillermo Suarez-Tangil, Juan Tapiador, and Pedro Peris-Lopez. "Stegomalware: Playing Hide and Seek with Malicious Components in Smartphone Apps." In: Department of Computer Science, Universidad Carlos III de Madrid. Dec. 2014. DOI: 10.1007/978-3-319-16745-9_27.

[26] *Types of Images*. URL: https://www.tutorialspoint.com/dip/Types_of_Images.htm (visited on 09/01/2020).

[27] *Unicode Text Steganography Encoders/Decoders*. URL: https://www.irongeek.com/i.php?page=security/unicode-steganography-homoglyph-encoder (visited on 07/20/2020).

[28] *What is steganography and how does it differ from cryptography?* 2019. URL: https://www.comparitech.com/blog/information-security/what-is-steganography/ (visited on 09/05/2020).

7 Eidesstattliche Erklärung

Hiermit versichere ich, dass diese Abschlussarbeit von mir persönlich verfasst ist und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wörtliche oder sinngemäße Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt für graphische Darstellungen und Bilder sowie für alle Internet-Quellen.

Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert versendet und gespeichert werden kann.

21.09.2020	[illegible]
Datum	Sally Addad