QR CODE STEGANOGRAPHY FINAL REPORT

By Kieran Carroll

C00241073

2021/2022

Y4 Cybercrime & IT Security

Institute of Technology Carlow

Project Supervisor: James Egan

Table of Contents

Table of Figures

Introduction

This document outlines the end product that was created as a result of implementing this project. An overview of the project is provided and the different functionality it offers is described and illustrated with screenshots. A review of both the aspects I failed to achieve alongside the aspects that I achieved is reviewed and compared to the initial features that were presented in the functional specification. The problems I encountered through the course of the development are also discussed. Finally, we review the tests that I carried to verify the quality of the tool's user interface and its products.

Project Description

Steganography was successfully implemented in this project through the use of the least significant bit method. This method works by taking advantage of the high level of redundancy that images have by swapping the least significant bits (the right most bit) in an image with the bits of our message, essentially embedding the message in a manner that does not visually change the image. By using this method, we achieve the goal of steganography which to provide secrecy by masking the fact that there any message present as the image does not appear to be altered in any way. The project focuses on the use of QR codes as a cover medium for steganography, however it also has the capability to use other images. The use of QR codes as a cover medium has some benefits over coloured images such as the being more size efficient as they are binary images which means many adjacent pixels have the same values which makes them especially compressible. The message encoded in the QR code, which is separate from the embedded message can also be used to deceive the adversary.

The Tool

The application provides the user with the capability to perform 3 core actions and provides a fluid interface for each activity. These 3 actions include generating a QR code, embedding a message and extracting a message. Additionally, the user can also choose to encrypt the message before embedding and decrypt it when extracting. The tool was developed as a windows desktop application using the Eclipse IDE and the java programming language. To develop the graphical user interface (GUI) the window builder plug-in was used in Eclipse. The ZXing library was used for the creation and reading of QR codes, and the bouncy castle library provided the necessary capabilities to perform AES encryption and decryption.

Application User Interface

Welcome

In this window the user is shown a welcome message and 3 buttons are provided for navigation to either the embed, generate QR or extract windows.

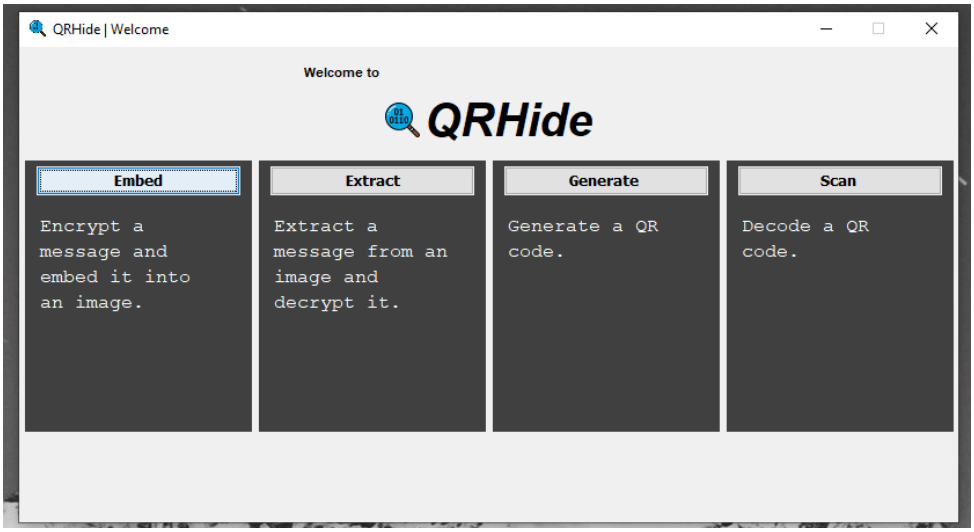

Figure 1 - Welcome Window

Generate a QR Code

On this page the user can generate a QR code and save it to their computer. They must enter a message which is capped 4,296 characters, this is the max number of alphanumeric characters a QR code can hold. The size the user wants the image to be must also be entered, the default is already set to 300x300, although a user can enter any size, they want up to 6000. Next the user should select the level of error correction that they want, which can be L, M, Q or H, the default is set to L. Finally, the user clicks the “Generate QR” button which the generates the QR and displays it to them the image box. The user can now save the QR code by navigating to File, then save where they can select a folder on their computer where they want it to be saved. The image must be saved in either the PNG or JPG image format.

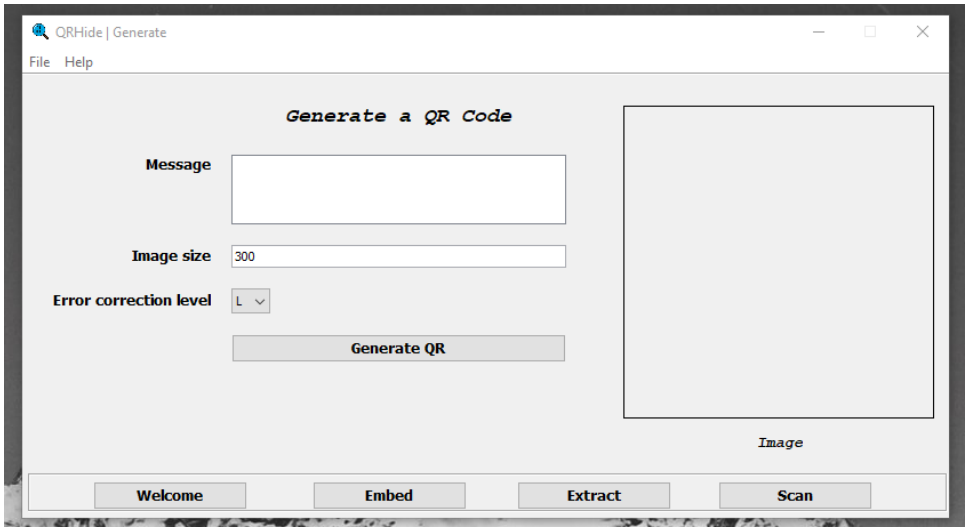

Figure 2 - Generate a QR Code Window.

Embed a Message

This page allows the user to upload a QR by navigating to file then uploading an image from their computer. The uploaded image must be in the PNG or JPG image format. The user then can enter a message of their choice and a key they if they choose to, which cannot be greater than 16 characters. The user then clicks embed which encrypts the entered message and embeds it into the least significant bits of the uploaded image. If the message is successfully embedded, then a message pops up that states that the message has been embedded. The user can then save the altered image to their computer by navigating to file then save. The image must be saved in either the PNG or JPG image format.

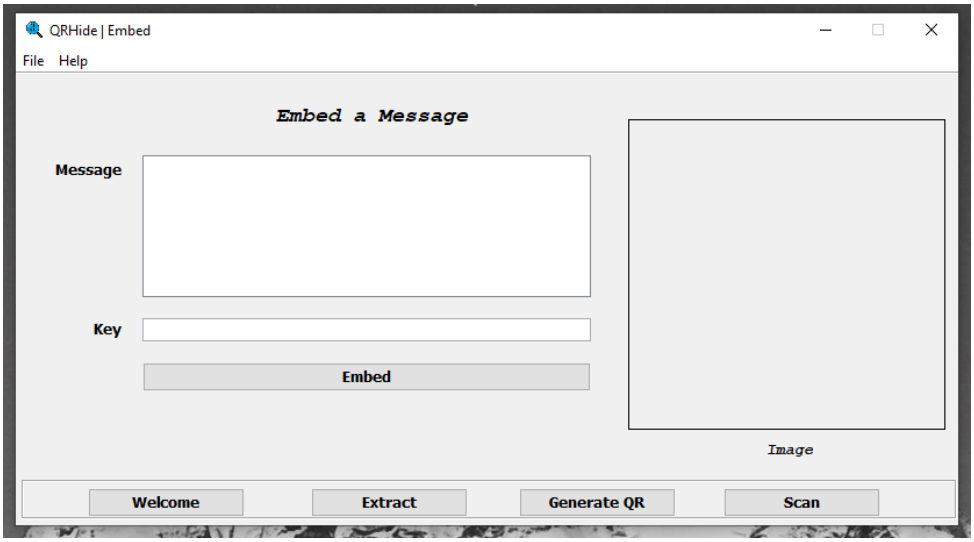

Figure 3 - Embed a Message Window

Extract a Message

The user is able to upload an image which is of JPG or PNG format on this page. The uploaded image is displayed in the image box. The user then clicks extract which takes the message out of the image and displays it in the textbox and a message pops up to say the message has been extracted successfully. If the message was encrypted, then the user must enter the same key that was used to encrypt into the key textbox. The user then clicks extract and the message is taken out of the image and decrypted, then displayed in the textbox.

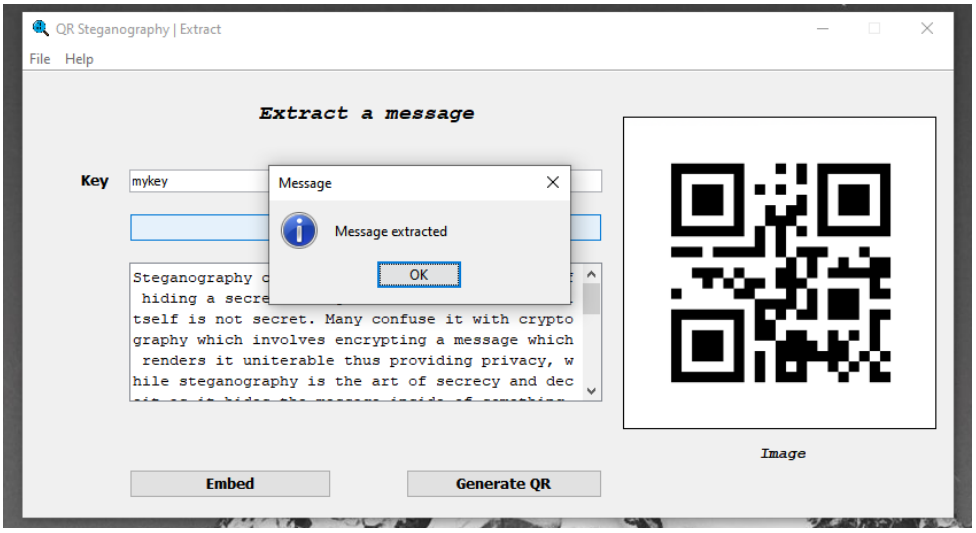

Figure 4 - Extract a Message Window

Scan

On this page a user can upload an image of a QR and scan it.

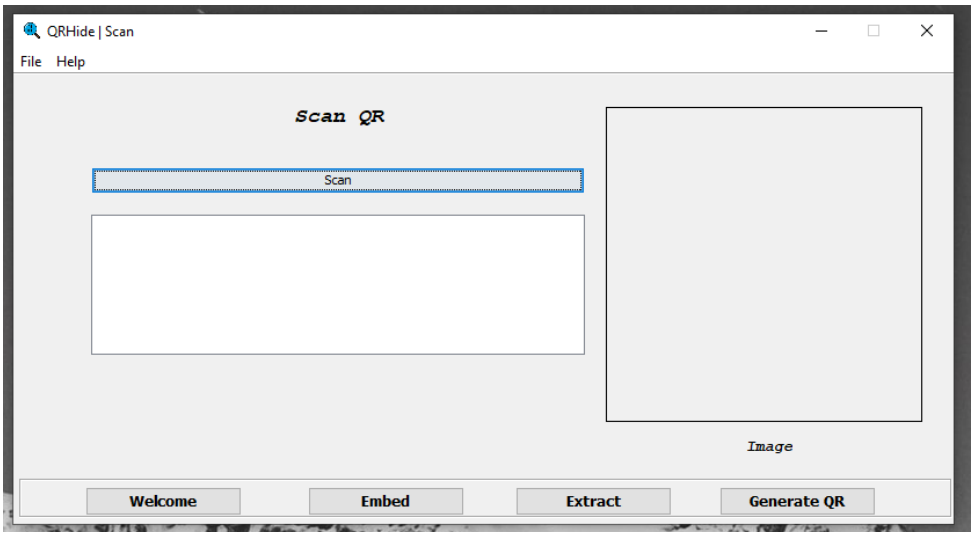

Figure 5 - Scan Window

Project Review

Through the research and development of this project I was able to achieve some aspects I had set out to do and also others that I was not able to implement due to several issues, this section of the document outlines these points and I then discuss what I would do differently if I were to start the project again.

Aspects achieved

I was able to achieve the following 6 core functions that I outlined in the functional specification:

1. QR code generation
2. Entering a message & key
3. Embedding the message
4. Extracting the message
5. Encrypting the message
6. Decrypting the message

From the start of the project these 6 aspects were the core pieces of functionality I wanted to have working by the end of the project, although I did have to revise and alter the way the steganographic method for embedding and extraction worked numerous times. I created a technique that successful embeds the message and extracts it. The final product provides a fluid interface which allows the user to easily perform all 6 actions.

Aspects not achieved

I was not able to fully utilize the capabilities that QR codes have. I wanted the message to be hidden inside of the QR code when it was being generated rather than in the image of the QR code. In doing this the QR code could be printed and a picture of the printed QR code could be used to extract the message. The tool I have created is not able to do this as the message is embedded into the image of the QR code rather than the data codewords, thus when printed onto paper and a picture is taken and uploaded to the tool, the message is lost. This problem is outlined in the testing portion of the document. It's noted that a QR that has a URL encoded in it can be scanned which takes the individual to a site that has the digital version, they can then upload the digital version to the tool and extract the message. It's a two-step process rather than one.

Problems Encountered

Creating the steganographic methods for embedding and extraction proved to be the hardest part of the project, especially because I wanted to make full use of QR codes as a cover medium. This meant exploiting the error correction feature that QR codes have. The initial method of embedding was supposed to work by replacing some of the data codewords when the QR code was being created with the bits of our message. Therefore, when the QR code is read then the altered data codewords are seen as errors and are corrected by the error correction feature, thus the message would be hidden in the QR code. To extract the message, we reverse the embedding procedure by reading the QR and after the masking step we extract the hidden data, followed by the original data which is recovered by the error correction codewords. Implementing this technique proved to be very difficult as it required that I create a modified version of a QR generator and reader, as they would have to both agree where this data is placed and where it ends when the QR code is created and read. Such a task was too difficult, and my technical ability limited me from being able to do this.

The second technique I tried to implement involved first generating a QR code, then an empty image that was equal to the size of the QR code. We would then insert the encrypted secret message into the empty image, the message would be scrambled all over the image. To embed the message, we would then XOR the empty image with the QR code. To extract the message the QR code would be read, then a second QR code would be generated with the same dimensions. The newly generated QR code would be XORed with the initial QR which would give us the scrambled image that has the secret message inside of it. Finally, the image would be unscrambled, and the message extracted and decrypted. I spent most of the time working on implementing this technique, especially the image scrambling portion which was the hardest bit as I had to educate myself on the topic of image scrambling, and how it is performed using chaotic maps, then attempt to translate that into java code. Overall, I wouldn't have been able to get the tool in a working state using technique as I would have run out of time.

To overcome the challenge of creating a steganographic algorithm with limited time, I created a method for embedding and extraction using the LSB technique. This involves embedding into the least significant bits of the image. Using this method, I was able to build the tool. However, implementing this method also wasn't straight forward as I encountered issues when creating the embedding algorithm as seen below noise can be seen in the image at the top of the image.

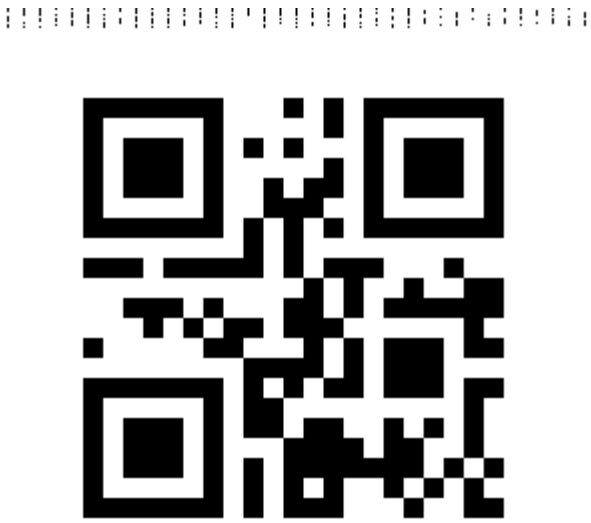

Figure 6 - Stego image with noise

The reason this happened was because I was using the buffered image type of byte binary, which means that there is one bit per pixel (either 1 or 0) thus this caused an issue because I was implementing it in such a way that would only work with images that have a bit depth of 8 or greater. Once I realised this, I change the image type which solved the issue.

What I learned

Through the research and development of this project I have learned a great deal about steganography, QR codes, Java programming and various other concepts. With regards to steganography, I had never heard of it before and from my research I gained in depth knowledge on the subject including the various techniques that can be used. I had to learn how QR codes worked, in particular the encoding and decoding procedures and how they would work as a container for steganography. My overall programming skills in Java also improved as a result of implementing this tool and I learned how to use the window builder plugin within the eclipse IDE to build the GUI for the tool. Through using the ZXing library I was able to learn how to build and read QR codes in Java. While trying to implement image scrambling in the project I had to read up on how image scrambling is performed using chaotic maps, especially ones that work with square matrices, this included the tent and baker maps. From this I had to understand the properties of chaos and how I can utilize it to generate chaotic keys that I would need for image scrambling. Investigating this niche topic allowed to me to gain an understanding of image scrambling and how it can be used lower the chances of

steganalysis attacks finding the secret payload. Besides the improvement of my technical skills improving, I also learned how to persevere when challenges arose.

What I would do differently

If I could start this project again, I would have changed my approach to tackling this project. I would have used python instead of Java, although I do not know python, I think it would not have been difficult to learn and it would have provided me with more libraries which would have made development easier. I also would've started by understanding how QR code work fully, before trying to implement anything. I would delegate tasks appropriately and finish one before moving onto the next one, I found myself jumping between the various tasks I had to do which led to confusion.

Testing

QR Code Testing

QR codes generated with the tool that contained a secret message were tested to see if the encoded message (not the secret message) could still be retrieved by scanning them with a smartphone both digitally and when printed on paper, in both cases they worked. A second test investigated whether the secret message could be retrieved after the stego QR code is printed out and a picture is taken of that QR code using a smartphone, then the picture is uploaded to the tool and an attempt to extract the secret message is made. The result was that the tool could not find the secret data, it was lost. It also noted that conversion between images that use lossy and lossless compression can in some cases also cause the data to be lost. Besides these two issues the secret message can be successfully retrieved with this tool from the stego image. It's noted that this tool works with both PNG and JPG images.

UI Testing

All functionality was thoroughly tested to check if the application works as intended. I tested the input fields to see what happens if the user enters the incorrect data type into a field, such as in the image size field on the generate QR window. When this happens, the application stops the user from entering the incorrect datatype. Characters restrictions on certain fields were also tested by entering the max

number of characters, the result was the appropriate error message was reflected back to the user. Where a user does not enter information into a text field that is required then the application would not proceed with the action and inform the user that information is required. Finally, if the user attempts to upload an image that is not of the PNG or JPG image format, then the application produces an error message back to the user. Overall, the UI works as expected and there is no apparent way to break the application.

PSNR

In this test we compare the difference between the original QR code and the QR code that contains the secret message using peak signal to noise ratio (PSNR). High PSNR indicates that the stego image is of good quality. The test results are outlined below. The result when PSNR is calculated between these two images is 84.962560980206973. It's noted that PSNR will decrease as the payload size increases. Additionally, by visually inspecting the two images there is no visual discrepancies between the two.

Figure 7 - Original QR

Figure 8 - Stego QR

Embedding Capacity

To display the embedding capacity that any image uploaded to the tool has I created a method that calculated the amount of secret data the image could hold. This worked by multiplying the width x height x 3. We multiply by 3 because 3 bits of our secret message can be stored in each pixel of the image. This gives us the number of bits of the secret message that the image can hold, to get the number of bytes we divide by 8. We minus 32 from this number because we leave the first 32 bytes of the image to store the length. The result of this is the number of characters the image can hold.

Figure 9 - QR Code

The embedding capacity of the above QR code image is calculated as follows 3x300x300 = 270,000 bits / 8 = 33,750 bytes – 32 = 33,718 bytes. This image can hold 33,718 characters of our secret message.

Conclusion

Overall, I consider the end product of this project to be a success. Although I was not able to fully utilize the physical aspects of QR codes as a container for steganography, I still hit all the main points I outlined in my functional specification, creating a fully functional application that allows the user to perform steganography and encrypt the payload.