Interim Specification

Steganography

Josh Ward

James Wilcox

17/02/2017

Introduction

Objectives –

* Research possible steganography techniques.
* Investigate the appropriateness of different file types for this application.
* Write a program to hide a user-selected image within a cover image.
* Write a program to retrieve the data hidden within an image.
* Implement encryption of the hidden data (optional).
* Create a program to detect the presence of hidden data within a file.

Steganography is the concept of hiding data in plain sight. Digital steganography is the successor to physical methods used in the past such as the use of secret inks to write messages on paper with otherwise innocuous messages written over them or writing messages on the scalp of a messenger which are then hidden by the hair.

In this project, a program will be created that will hide information within the user’s chosen image file. The program will then also be able to retrieve this information and save it separately in the desired file format of the user. If time permits, encryption of the data to be hidden will be investigated and the creation of a program to search for hidden data will also be considered.

The most appropriate file type for this application is the .bmp file type. The .jpeg file type is unsuitable for this application due to its aggressive compression of files. The use of .jpeg would likely cause the data hidden within the file to become corrupted or unrecoverable. There are some other file types that could be used for this application such as .tiff and .png files, however, in addition to the RGB values required for the image analysis, these file types also contain an alpha value. This alpha value is responsible for dictating the opacity of the pixel, this extra information would increase the complexity of the information to be hidden, making the embedded data more vulnerable to discovery.

Various stenographic techniques are used for hiding files within images, these techniques include; least significant bit steganography is the simplest and most common stenographic technique used for image steganography. The technique works by taking a cover image and hiding another Image within the least significant bits of the pixels of the cover image, this in turn creates a new stenographic image which contains both images however only one is visible to an observer. However, this method is susceptible to a “visual attack” where upon a close examination an observer can notice the noise in the image, making it the most exploitable method.

Another method, which can be used, for image steganography is discrete cosine transform steganography, this method works by taking a JPEG image and separating the image into parts of differing importance, this separates the image into high, middle and low frequency. The hidden image is then hidden within these frequencies; this allows the image to be hidden and is more effective than LSB steganography as it is not exploited using a “visual attack”.

Program Function

Figure 1 below shows the proposed class hierarchy for the program.

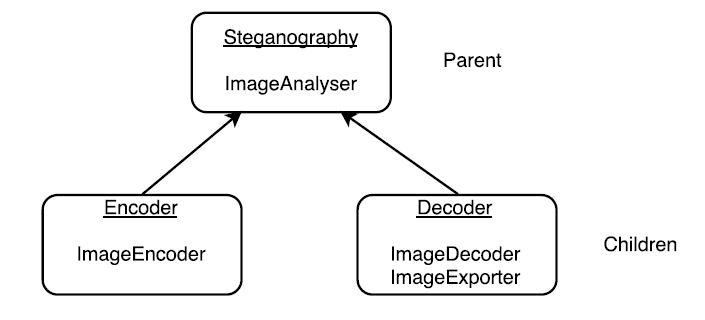


Figure 1 Program class hierarchy

A brief outline of each function detailed within Figure 1 can be found below:

* ImageAnalyser – Converts a 256 colour bitmap into an RGB sequence to be stored in a multidimensional array.
* ImageEncoder – Hides the binary sequence of the data to be hidden within the two least significant bits of the cover image.
* ImageDecoder – Retrieves the colour information of the hidden image from the stego object and adds it to an array.
* ImageExporter – Converts a binary sequence into a 256 colour bitmap image.

Both child classes (Encoder and Decoder) inherit the ImageAnalyser function from the parent class.

The process of hiding the image within the cover image is controlled by the Encoder class. A rough outline of the process that this class will follow can be found below:

* Create a bespoke function called ImageAnalyser which calls a ‘carrier’ image and an ‘outfile’.
* Two instances of this function will be created, firstly for the cover image and then for the image to hide.
* Open image and convert to RGB.
* Get dimensions for image (width and height).
* Verify that the cover image is four times the size of the image to hide.
* Create array and store pixel RGB information within it (empty initially).
* Create a loop to get RGB value of each pixel for x and y within image dimensions.
* Add each RGB value to array by using append function.
* The ImageEncoder function will then be used to convert the RGB values of each pixel to a binary pickle stream.
* The data from the pickle stream of the image to hide is then hidden within the two least significant bits of the pickle stream of the cover image.

Figure 2 below shows the proposed flowchart for the Encoder class.

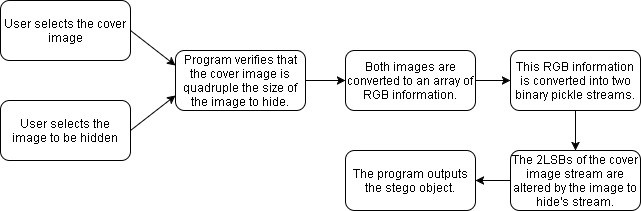


Figure 2 Encoder class flowchart

The Decoder class will be responsible for taking the stego object produced by the Encoder class and retrieving the data that has been hidden within. A rough outline of this process can be found below:

* When the user wishes to retrieve the hidden data from the cover image, they will create an instance of the Decoder class.
* The user will then be prompted to enter the name of the stego object file and the name of the file they wish to save the hidden image to.
* The selected image will then be analysed with the ImageAnalyser function to create an array containing the R, G, and B values of each pixel of the stego object.
* The ImageDecoder function will then be used to convert this array into binary and select the last two significant bits of the colour information of the stego object and retrieve the data hidden within these bits.
* The values of the hidden image are then brought back together to form the original colour values of the hidden image and stored in a binary array.
* The ImageExporter function is then used to convert each value in this array back into its RGB equivalent and this information is used to rebuild the hidden image.

Figure 3 below shows the proposed flowchart for the Decoder class.

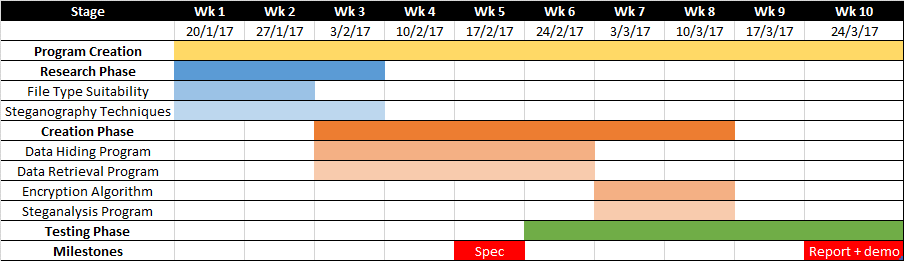


Figure 3 Decoder class flowchart

Project Timeline

A Gantt chart was created and can be found below in Table 1 detailing the stages in the project and their target completion dates.

Table 1 Steganography Program Gantt chart



The tasks detailed above were split up as follows:

Josh Ward: Steganography Techniques, Data Hiding Program, Encryption Algorithm.

James Wilcox: File Type Suitability, Data Retrieval Program, Steganalysis Program.

Combined: Testing Phase, Spec (Interim Specification), Report + Demo.

Software Testing Procedure

Stress testing – Tests will be done to investigate the programs ability to function when increasingly larger file sizes are input into the program. This testing will be run to the point where the software fails such as a crash, or the program begins to function incorrectly. An automated software testing package would be preferable for this application such as IBM’s Rational Function Tester. This forms the main section of the White-Box testing.

Black-Box Testing - It is proposed that a member of the public is asked to use the program and give feedback on the intuitiveness of the application from the perspective of someone who has no knowledge of the functioning of the program.

In addition to testing the functionality of the program, tests could be run to compare the program to existing solutions currently available online. This could be used to test both the effectiveness of the encoding process as well as the ability of the decoding process to retrieve the information accurately and efficiently.

Scope and Limitations

There are several additions that could be made to the program if further time was available such as offering multiple steganography techniques such as sine wave manipulation in addition to the LSB method used in this program. Due to the time constraints and lack of experience with creating programs, a relatively simple method (Least Significant Bit) has been selected.

Another area of improvement would be to create a GUI for the user to interact with. This would make the program much more intuitive to interact with as the user is likely to be more familiar with using a user interface than the python shell.

A large limitation of this program is that it is limited to .bmp files due to the constitution of other file types. However, given more time, these conflicts could be investigated further and solutions could be found.

If further time was allocated a cryptographical random function aka scattering could be investigated to decrease the ease with which the message can be decoded by an unintended recipient. This function randomly hides the data within different pixels of the cover image as opposed to in a predictable, uniform spread.

References

* Ways of detecting steganography <http://security.stackexchange.com/questions/2144/detecting-steganography-in-images> (accessed on 28/1/17).
* Information about steganography <https://en.wikipedia.org/wiki/Steganography#Citations> (accessed on 28/1/17).
* Information about detecting steganography <https://en.wikipedia.org/wiki/Steganalysis> (accessed on 28/1/17).
* Flowchart information <https://en.wikipedia.org/wiki/Flowchart> (accessed on 29/1/17).
* Pseudocode information <https://en.wikipedia.org/wiki/Pseudocode> (accessed on 29/1/17).
* Steganography example – \*Login required\* <https://www.packtpub.com/mapt/book/networking-and-servers/9781784392932/6/ch06lvl1sec52/hiding-a-message-using-lsb-steganography> (accessed on 29/1/17).
* Software testing information <https://en.wikipedia.org/wiki/Software_testing> (accessed on 2/2/17).