Image Encrypt Project

Final Project for CS 4003 Fall 2017

Nicholas Maltbie

Introduction

This project will use haskell to encrypt files, text, and images into other images using steganography by modifying bit values for images. In order to do this, I will utilize a few libraries to edit the images and encrypt values from files.

This project utilized functional programming because haskell is a functional language. I have created applications similar to this in Java and Python but this is my first time doing this application in a functional language. My original goal was to implement different kinds of steganography but learning and creating an application in haskell for basic Least Significant Bit (lsb) steganography in images was the only part I could complete with certainty that it worked.

In order to better understand steganography as a field, I looked into C.P.Sumathi, T.Santanam and G.Umamaheswari's paper "A Study of Various Steganographic Techniques Used for Information Hiding." This gave me a good starting point to identify and look into different kinds of steganography.

My source code for this project can be found at https://github.com/nicholas-maltbie/Image-Encrypt. This project has the source code and compiled code for Ubuntu os. The files that are relevant are **encode.hs** and **decode.hs**. These have the ability to encode a file (of any type) into an image or decode a file from an image.

Usage

To use the program, compile and then run encode.hs. This will prompt you with (1) the file to save in the image, (2) the image to save it into, (3) the file format for the result, and (3) the name of the output image.

To decrypt information from an image, compile and run the decode.hs. This will prompt you with (1) the image file with hidden information. Then the program will read the hidden file from the image and prompt the user with (2) the name to save the hidden file out to.

Description

When hiding information in an image, the program will generate a new image file that has the information of the file hidden in it. The information can be hidden at different densities in the image. With lsb, the program will use the least significant bits in the color channels for the pixels to hide the information. There are up to 8 bits in each pixel color channel which means up to 3 bytes can be stored per RGB8 pixel but the more bytes that are used the more distortion is created.

The program first hides the length of the file, then after that writes the file name and the file data into the image. This data is hidden in the image well and, although it causes distortion, is almost invisible to the human eye.

When given a file and an image, the program will decide how many bits per color channel should be used to save the information in the image. The program will select the least bits possible as to cause the least distortion and then splay out the changes across the image to make the changes difficult to detect. If the program distorts more than half of the pixel (distortion level of 5 or greater) it will sometimes be unreliable as the png compression format does not allow very high distortion. The program will work with any level of distortion when exporting to bmp format as bmp has no compression.

Challenges

One of the main problems with this program is that the image cannot be compressed when the file is written out to the computer so file formats like JPG can cause problems since they have lots of compression. Additionally, when information is hidden into an image, compression methods that image formats like PNG use are less useful and cause the file size to become larger. I have sampled some of this in the results section of the report.

Additionally, when hiding the pixels into the image, it is difficult to determine where to put the pixels. If all the pixels are stored in order it is very obvious that the image has been distorted, although if they information is spread out too much it will be difficult to decipher. In order to resolve this, I splay the information evenly across all pixels and color channels evenly to make the distortion seem minimum. This does not hide the information any better but also makes it harder to detect the distortion by looking at the image.

Results

Below is an image with a test string encoded into it.

Fig 1. Original

Fig 2. With test string

The two images look identical and no distortion can be detected by a human eye. The original image is 898.1 kB, and the image with a test string is 898.7 kB. This is a complicated image and the test string was only 50 characters long so there isn't much added to the file size.

Fig 3. Alice In Wonderland Image

Above is the same image with the raw text of *Alice in Wonderland* hidden in the pixel data. There is still little to no distortion. If the contrast on the image is significantly increased, some of the solid blue sections of the sky would look different than that of the original image with more noise in sections with solid color. Still, since only one or two bits per color channel is changed a human eye can not see any difference.

The text of *Alice in Wonderland* is 167.5 kB, the original image is 898.1 kB. The image with *Alice in Wonderland* hidden in it is 1.4 mB large. This is a significant increase in file size. This image size increase is much more pronounced in images which can be compressed a lot due to

large sections of the same color. When data is encoded into an image, it adds noise to the picture and makes it much more difficult to compress to a smaller file size.

Fig 4. Tree Picture

Fig 5. Tree with resume

Just to show off the program, I also encoded a pdf file into an image because it is not limited to plaintext. This works very well and since pdf files are very compressed it can be done with and only minor distortion in the image. This shows off how any file can be stored in an image and not just plaintext. This is very useful and means that a user could encrypt a file and then hide it in an image. This would allow sending of encrypted, hidden messages in images. This can be very useful if people might guess that information is being hidden in the images.

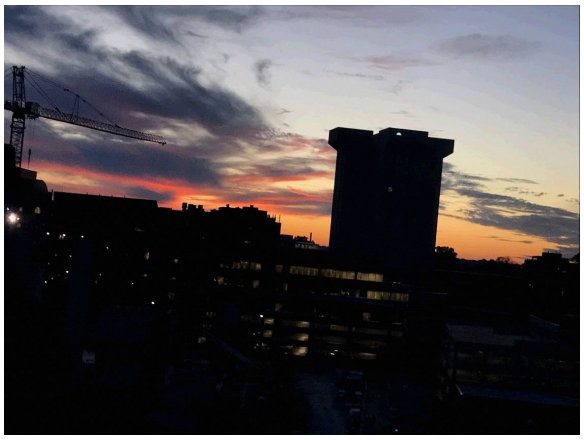

Fig 6. Image within image

Fig 7. Tree image

Since image files are also ‘files’, an image can be hidden within an image. Since the file has some compression it leads to interesting situations. For example, a file is smaller due to compression so Fig 6 is the original image (Fig 1) hidden in itself. There is some visible distortion in the clouds but it looks passable to most people. The image on the left above, Fig7, is the image of the tree from Fig 4 but with another image hidden within it.

Hiding files within images causes the file size of the image to increase as compression is not as effective and the image still needs to maintain all the original pixel data so the information can be restored.

Fig 8. White test image

In order to investigate how much the image is distorted, I encrypted a text file into a plain white image so the distortion would be as obvious as possible. The leftmost image of Fig 8 is the original white image, the middle image is the image with the small text file hidden in it, the rightmost image is the image with hidden information but with raised contrast and decreased brightness. The third image shows the distortion of the pixels splayed across the image. The distortion is nearly invisible; although the file size does increase from 300 bytes to 900 bytes (the text file was only 50 characters). This clearly shows how the distortion is nearly invisible.

These results seem promising and show the strength of the least significant bit steganography. Implementing this in haskell and using functional programming was very difficult but possible through use of external libraries.

Conclusion

Although I did not achieve my original goals of multiple methods of steganography, since I can encrypt any kind of file into an image, this program also works with encrypted files. Simply encrypt the data before hiding it into an image and then send it over open channels. The advantages of stenography mostly rely on the fact that observers don’t know that hidden messages are being sent. I could post an image with hidden instructions onto social media and people who know to look for the message will find it while others don’t even know that a message is being communicated.

This exercise of completing the program in haskell was challenging since processes such as editing an image are mure more easily defined in a procedural manner instead of functional programming. The program works well and could still be improved but I’m happy to report my progress and have solidified a great deal of what I was taught in the course.

References

C.P.Sumathi, T.Santanam and G.Umamaheswari. "A Study of Various Steganographic Techniques Used for Information Hiding" (https://arxiv.org/pdf/1401.5561.pdf)

Rajkumar Yadav, Ravi Saini & Gaurav Chawla. "A Novel Approach For Image Steganography In Spatial DoMain Using Last Two Bits of Pixel Value" (http://www.cscjournals.org/manuscript/Journals/IJS/Volume5/Issue2/IJS-77.pdf)

Mamta Juneja and Parvinder Singh Sandhu. "A New Approach for Information Security using an Improved Steganography Technique" (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1002.2544&rep=rep1&type=pdf)

Shamim Ahmed Laskar and Kattamanchi Hemachandran. "Steganography based on random pixel selection for efficient data hiding" (https://www.academia.edu/3216126/STEGANOGRAPHY_BASED_ON_RANDOM_PIXEL_SELECTION_FOR_EFFICIENT_DATA_HIDING)

Matteo Fortini "Least Significant Bits (LSB) insertion" (http://www.lia.deis.unibo.it/Courses/RetiDiCalcolatori/Progetti98/Fortini/lsb.html)