C Coursework: Steganography (Hardware Software Interface F28HS2)

08/03/2018

By Mark Schmieg

(H00238262)

BSc Computer Science at Heriot-Watt University in Edinburgh

This coursework is about implementing steganography. To achieve this, I have created three structures:

- A “LinkedList struct”, to store all the comments.
- A “colours struct”, to hold the different pixels.
- A “PPM struct”, to contain the ppm image.

GetPPM

I started off by implementing getPPM, where the program goes through the whole image and gets the individual parts and stores them into the “PPM struct”. The function getPPM starts off, by getting the ppm format, which in this coursework is P3, then it checks if the format is met or not. If the format is incorrect, it will print an error message. If the format is correct it will proceed to store the comments in a “LinkedList struct”, which is then put into the PPM struct. Then the width, height, and max value are also stored into the “PPM struct” as integers. For the pixels, the function goes and takes 3 bits at a time, which represent the 3 bits (red, green, blue) of a pixel, and stores them into a “colours struct” which is composed of 3 integers. Then that colours struct is stored into an array of colours structs, which is held in the “PPM struct”. For the pixels, I have chosen to make an array of structs as this is faster to access, since you can access any point in the array in O(1) time, and because it is also easier to visualise. At the end, the function returns the “PPM struct”.

ShowPPM

In the showPPM function, the program just outputs the “PPM struct” that it receives. It first prints the format (P3), then the comments, after that, it prints the width, height and max value. At the end it prints all the different pixels, every pixel is at a new line so that it’s easier for the user to see the different pixels if they choose to open the PPM, and it’s also to have a persistent format.

The showPPM has the following format:

```
P3
#Comments
....
r1 g1 b1
r2 g2 b2
….
```

Encoding

To encode, I have made random number generator that generates a random number, from which the modulo by the size of the PPM divided by the length of the message -1 is calculated, so that the whole message can be inserted into the PPM image. When going through the message, the function checks whether the encoded value is greater than the max value, if so it will throw an error. For the pixels that get a letter inserted, if the value of the letter is equal to the red bit value, then the green value of that pixel gets 5 subtracted from it, so that the decode will know, what red bit has been changed, if the values are the same, and avoiding making the green bit greater than the max value. Otherwise, the red bit gets replaced by the letter value. Without the original ppm image, a person cannot start any decryption, because this encryption has no real pattern, which can be followed through when only having the encrypted version. Initially, to encode it, I divided the red colour by 3 and added that value to the value of the character, but that approach had issues, because I didn't handle the situations where the red value would be equal to the encoded value, and where the encoded value would be greater, than the max value. I changed my code to the current approach as it is easier to handle those errors and it is more efficient.

Decoding

To decode, the program goes through the whole encoded ppm file and checks if the red and green bits are different from the original ppm image. If they are the same it just goes to the next pixel, otherwise it saves the value of the red bit, to in the end display the secret message encrypted in the image.

Main

For the main method I have a general argument check to check that the number is not greater than the one required for decode and less than the one required for encode.

Then I have also implemented a few file checks to make sure that the files being opened are not empty, and if they are, the program will throw an error.

To encode run the following code:

./steg e file1 > file2

To decode run the following code:

./steg d file1 file2

Conclusion

Getting the most efficient and easiest approach to encode and decode a ppm, was the key of this coursework. Encoding a message into a ppm image might also not be the best idea to send a secret message, as big messages can be spotted on the image as many pixels are altered, small messages however are very efficiently hidden in the image. I found this coursework very challenging, but I learned a lot about coding in C, an about an interesting encryption method.