RGB Herring

James Bors, Matthew Sutton, and Calvin Crino

***Abstract*— By taking in a file that is a picture we show how one can use steganography to encode or decode. Depending on the users specifications the program can be given the key which is the particular encoding sequence and a file, which will decode the file and output the message on screen. Adversely it will intake a file and with the users choice encode to specification. (If and only if the message is not too large)**

I. INTRODUCTION

A picture may be worth a thousand words and in some cases so many more. Think about the last time you looked at a picture and the emotion it exhibited. Hope, strength, wonder, the descriptions can be endless because pictures can express many things but consider the picture was intended to mislead. How is one supposed to know the picture was misleading?, the frank answer is you are not. Steganography is the process of concealing information within non-secret text or data. Thus steganography can be used in conjunction with a picture to make a misleading picture. The misleading picture can be of anything or anyone but the real trick is for no one to notice that it is not actually just a picture. Using different techniques expressed within the methodology section of this paper we show how one can manipulate a bitmap picture to hide secret text or data within it. Due to the different variations in each picture format the steganography that we portray is limited to bitmap pictures.

II. BACKGROUND

A. STEGANOGRAPHY

The concept of steganography is not a new one, actually it can be traced back to the ancient Greeks according to Rebah, Kefa (Kefa, 2004). Kefa explains in his article that secret "... messages were either etched into wooden tablets or even tattooed on a messengers head" the messenger then would shave his head to reveal the message (Kefa, 2004). The technique is that Kefa explains seems ridiculous to one extent because the messenger would have to live with a tattoo for his entire life, but concealing information in that time could have been very challenging. Concealing information in our era is much easier with the amount of mediums at disposal. Pictures, Text, Software, and even Music can be encoded with secrets.

B. Encoding Theory

Encoding for our purposes is this process of manipulating one input such that output contains the input but significantly transformed so that without knowledge of the method deducing the input is difficult;

$$f(x) = x + 2$$

Consider this function above where it takes in a number x and modifies it by 2. This function is similar to what we are doing to an extent. The program we have designed takes in an input, which in our case is a file that holds a picture. Next, we take this picture and slightly modify it as a function would and return the result to the user. There are plenty of different ways to encode a file by using different functions as explained above but we will particularly be describing 4: Monochrome LSB, Chromatic LSB, Keyed Monochrome LSB, Keyed Chromatic LSB.

C. Decoding Theory

Decoders work in reverse the encoding function. So using the same example the inverse function f' of f is:

$$f'(x) = x - 2$$

The program we have designed takes in picture and asks the user how it was encoded. Depending on the users' request we can try to decipher the encoding that was placed on the file at hand. It it very important that the user knows how the file was encoded, or decoding a file in a wrong decoder will give you nonsense information.

D. The Bitmap Format

In order to understand the different ways to encode the picture we must first explain the necessary information needed. Extracting the picture's details are made readily available from the BitmapHeader. This concept of finding everything inside the BitmapHeader wasn't known but it only took a few searches to obtain the information. Starting with the original bitmap picture which was intended to run on Microsoft version 1.x we can see how pictures have evolved with the evolution of operating systems (Murray and vanRyper, 1996). The encoding sequence that we use works for bitmap pictures of version 4.x and of lower compatibility. Murray and vanRyper explain Bitmap versions of 1.x-4.x as if it were implemented in a c++ program; the first section defined is the File Header in every version (Murray and vanRyper, 1996). The information associated with the File Header that we are using is: Size, Width, Height, BitsPerPixel, and SizeOfBitmap; these are described on the image below version 4.x on Murray and vanRyper's "Microsoft Windows Bitmap File Summary"(Murray and vanRyper, 1996). Referring to the first variable that is necessary is the size which defines the size of the header in bytes which must be saved in order

*This work was supported by The Pixel Bandits

to traverse the picture code to find the start of the pixel array. Next we need to know the width of the image (in pixels) which will be used to format the width of the image if needed. The height of the pixel map in pixels is needed to know how many rows are going to be traversed within the encoding loop. The BitsPerPixel has to be 24 in order for the encoding sequence to work, if the data states otherwise the encoding will halt and return to the main menu. Lastly we need to retrieve the SizeOfBitmap which refers to the picture array where the picture data is stored.

E. Monochrome LSB

For the Monochromatic LSB encoding method, we take in a color from the user to encode all of the message bits into. The color code gathered is loaded into the program, the first bit of the message is encoded, three bits are skipped and the process repeats. This leads to a sparse encoding across the image, ensuring that the maximum image clarity is preserved. Unfortunately this leads to smaller encode-able messages and less security for more sensitive information. A prying eye only requires 3 attempts to guarantee a successful decode when this method is used.

F. Chromatic LSB

encoding one LSB per pixel as in Monochrome leaves the image minimally changed, however if information density is favored then Chromatic LSB is preferable. Chromatic LSB's main operation is replacing the LSB of each channel of a pixel. this means that Chromatic LSB has three times the density as the same image with Monochrome LSB. to implement Chromatic LSB we perform the same operation as monochrome LSB but count by 1 instead of 3.

G. Keyed Monochrome LSB

This method is an extension of the Monochromatic LSB with a few very important distinctions. First being, the color in which the message bits are stored into are randomly chosen for each new pixel. While this still leaves the message sparsity with its positives and negatives (namely pixel clarity and message length), the security of the information increases dramatically. To ensure a secure encoding sequence, the user provides an 8 character long password to "lock" the picture with. This password is used as a seed for the java pseudo-random number generator used. To encode the message, the program generates a 0, 1, or 2 for each new pixel it encounters. This value corresponds to the color used (R, G, or B).

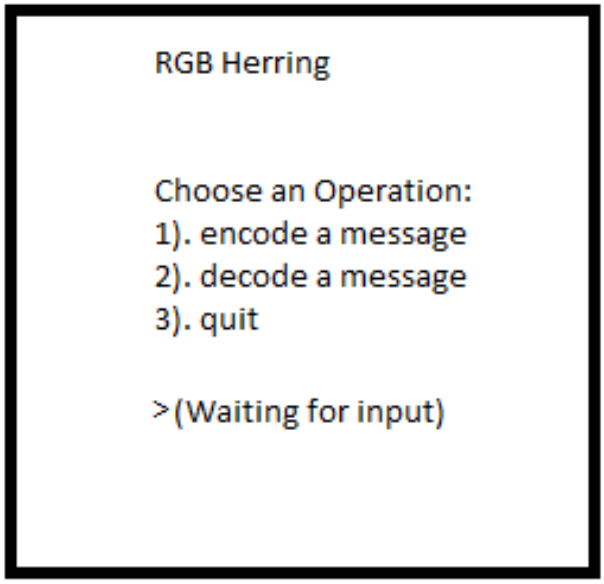
H. Keyed Chromatic LSB

Keyed Chromatic takes the methodology of Keyed Monochrome but adds additional complexity to further increase security and density. In Keyed Chromatic a key acts as the seed to a pseudo-random number generator. Then at each pixel the next random is used mod 6 to generate a value 0 to 5. Instead of choosing the red green or blue channel, the random is used to choose a permutation of rgb, for example gbr, and then the next 3 bits are encoded to/decoded from those channels in that order. for example to encode the character 'A' the pseudo-random sequence might generate 3,0,5 which corresponds to the permutations gbr, rgb, bgr so then the character value A, 01000001 would be stored in the lsb's in this order 001000-10. notice the skipped space which would be the first bit on the next character.

III. Methodology

A. User Input

The particular input to the program has to be exact. When running the program the user gets asked whether they would like to: (1) encode, (2) decode, (3) quit. Depending on the input the program, being designed in mips, will jump to a subroutine where the program will start executing the next set of instructions. Considering the user has not chosen option 3, we will ask the user for a path to the file they are using in the program. The user is given a sample script to show what it should look like. We used "Enter the full file path to the Bitmap: Ex. C:/Users/JohnDoe/Desktop/project/picture.bmp" as our example image. In this example picture.bmp is the bitmap picture that is being manipulated.



B. Encoding

After adding the correct file path to a picture that is located on the users C Drive we can give the user the option of which encoding algorithm they would like used. It is very important that the user remembers exactly how the picture was encoded or information could be lost in the translation process. The encoding algorithms we have described earlier are the ones present in the option-list including the most basic case which is chromatic LSB encoding. After the user chooses their intended option we must ask the user for a path to save the file to, this could look like "C:/Users/JohnDoe/Desktop/project/secretMessage.bmp".
The file will show up in the folder project, which is on your

desktop in this particular case, and you will find the file secretMessage.bmp.

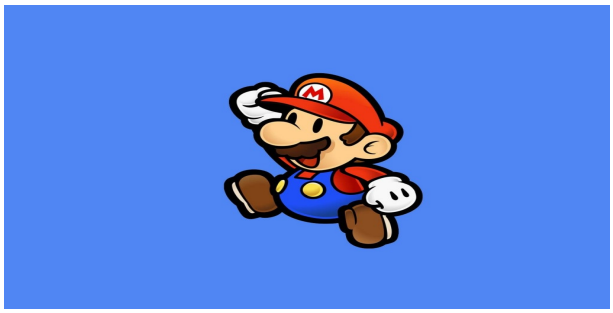
C. Decoding

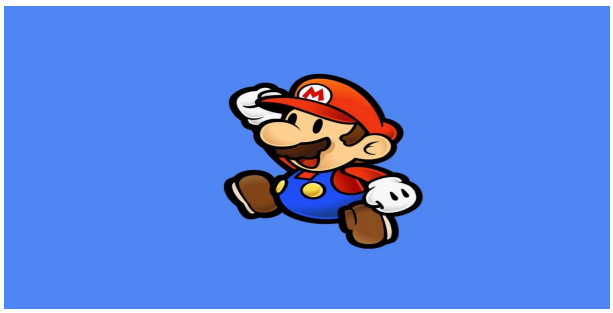
When given the correct path to the file the user needs decoded a new set of options will arise. Depending on the encoding sequence the particular file that has been uploaded has, should reveal which option to choose. An example would be uploading a file that used the Chromatic LSB and selecting option (2), which is ”monochrome LSB”. The program will jump to the specific decoding instructions we have created inside of mips to execute the users selection. The hidden data will show up in the Run I/O of the mars simulator. If the user incorrectly chooses the decoding sequence, or enters the wrong password if using a keyed method, data will still show up in the Run I/O but the sequence of letter or integers will be undecipherable.

IV. Results

A. Original vs Encoded

By using encoding sequences that do not directly change an entire sequence of color we can hardly see the changes directly in the encoded picture vs. the original. Considering there are hundreds if not thousands of ways to write an algorithm for a different encoding sequences we believe that our algorithms work wonders. Below we show an original vs. an encoded picture to show the differences.





While the images are pretty small there is no noticeable difference from image 1 to image 2. No noticeable differences are exactly what we are looking for. When creating an image that has hidden text within it the last thing that we are trying to do is give the reader a sense of what is happening in the background. Considering the reader has no idea that the picture has been encoded an average user would just view the picture and move on.

B. Seeing Nonsense

As a result of having simple encoding it will be easy to decipher a code once the user finds the pattern for how the data was encoded. In order to increase the security, in some of our encoding techniques we require the user to give the key to the encoded sequence in order to unravel the text in such a way that is readable. Using a wrong key to the right encoding sequence will give you nonsense instead of the ”secret message” you might have been looking for. The separated images below have been encoded using the Keyed Chromatic LSB method. We show here incorrectly entering the key vs. correctly entering a key.

```
Choose the Encoding key:
1).chromatic LSB
2).monochrome LSB
3).keyed monochrome LSB
4).keyed chromatic LSB
0). abort

>4
Enter the key:
        Max 8 characters ex: "ApplePie".

>wrongpas
RT©□!j□!l□9□ãn±òbc©□×mr¨uxV$□ÿÿÿÿÿÿÿÿÿÿ

RGB Herring
```

Above is what you will see if you enter the wrong encoding sequence or the wrong password when you decode. The gibberish on the line above ”RGB Herring” is the result. Below is the expected result when the correct password is entered.

```
Choose the Encoding key:
1).chromatic LSB
2).monochrome LSB
3).keyed monochrome LSB
4).keyed chromatic LSB
0). abort

>4
Enter the key:
        Max 8 characters ex: "ApplePie".

>password
This is the correct output.


RGB Herring
```

Even though we picked the correct decoding sequence (”Keyed Chromatic”) providing the wrong key will not give you any useful information. Referring to the first image shown above we can see how some of the characters look completely random. It has to be noted that this is not precisely what a wrong key would look like every time, because for every image you try this on you will receive a different result. This image only shows one example of

how badly our program spits out the encoded message when a wrong key is entered.

C. MIPS performance

The size of the image will directly affect the time it takes to execute the encoding. Frankly speaking a smaller picture will have a smaller pixel array which directly makes the encoding sequence execute faster because it is less data to parse through. The times may vary slightly from one encoding algorithm to the next but the bulk of the time spent is in reading the array. We also noted that because we are in MARS and we are entering text through a prompt and not through .txt file input, the maximum number of characters we chose to encode is 2 to the power of 16, or 65,536. This is equivalent to around 13,000 words. This number can be increased to the maximum MARS allows if desired, and then the number of characters encoded would purely depend on the size of the image. Adversely to the encoding sequence, the decoder relies purely on math and because it is not manipulating anything the execution time is nearly instant.

V. Conclusion

Originally most of the team members could only imagine what a picture with a hidden text would look like. "It will be very blurry in areas",or "it will show clear color transitions that should not exist" were just some of the original thoughts. Thus, this lead us to our first task within the project learning how to read a bitmap picture to so we could manipulate it. When trying to get resources for how to extract the information from the bitmap header it was nearly impossible to learn the process. Most of the resources that are readily available are written in c++ which were not very helpful for us. It was not long before we learned the sequence of the Bitmap Header which lead us to task 2, which is manipulating the array. Task 2 required a little bit of theory thinking about how changing particular bits will adversely change an image. Our research led us to understanding different bit depths sequences for Bitmap Pictures. We decided we would first learn the process of manipulating a 24-bit depth Bitmap picture. We choose this because of the unique structure its color array portrays. Adversely to the 24-bit depth the other depths tend to fluctuate drastically in their structure. Thus we believed if we could learn one bit depth pattern we believed we could succeed in the others. The structure we are dealing with for 24-bit pictures is RBG 8.8.8 where 8 bits represent each color respectively. Using this array information we choose a particular algorithm (1-4) and manipulated the bits to hide the message respectively. We had been practicing user input on MIPS and transferring data to and from registers so we had some idea on how this would work. All in all we were impressed on how subtle a picture can change with so much text comprised within.

A. To Pixel Bandits and Beyond

The Pixel Bandits had high hopes of creating a file encoder/decoder that can work across all platforms of bitmap pictures. If we were to advance our code in the future we would probably start at bit depth 32 and then work our way down to 1. In addition we thought about the idea of having a way to save the hidden message in a file after you have decoded the picture. Currently the program will only allow you to view the hidden message if you are running MARS because that was how it was designed but the problem is you cannot copy anything from the output log in MARS. Thus if you were in a hurry you could not read the message on the go. We would like to note that no pixels were harmed within the making of this program.

ACKNOWLEDGMENT

The Pixel Bandits would like to thank, MarioMayhem.com for the great bitmap picture of Mario. We would also like to thank Nintendo. The image was fun to encode due to it was a huge part of our childhood. Thanks again.

References

[1] Rebah, Kefa. Steganography-The Art of Hiding Data. Enzymes Application in Diagnostic Prospects, Academic Journals Inc., USA, 2004, scialert.net/abstract/?doi=itj.2004.245.269

[2] Murray, James D, and William vanRyper. Microsoft Windows Bitmap File Format Summary. Microsoft Windows Bitmap: Summary from the Encyclopedia of Graphics File Formats, O'Reilly, Apr. 1996, www.fileformat.info/format/bmp/egff.htm

[3] mariomayhem.com