CPSC 350 Data Structures:
Image Steganography

Nick Nabavian
nabav100@chapman.edu
Nov. 28, 2007

Abstract: Steganography is the process of hiding a secret message within a larger one in such a way that someone can not know the presence or contents of the hidden message. The purpose of Steganography is to maintain secret communication between two parties. This paper will show how Steganography is used in a modern context while providing a practical understanding of what Steganography is and how to accomplish it.

1. Introduction

Steganography is the process of hiding a secret message within a larger one in such a way that someone cannot know the presence or contents of the hidden message. Although related, Steganography is not to be confused with Encryption, which is the process of making a message unintelligible—Steganography attempts to hide the existence of communication.

The basic structure of Steganography is made up of three components: the "carrier", the message, and the key[1]. The carrier can be a painting, a digital image, an mp3, even a TCP/IP packet among other things. It is the object that will 'carry' the hidden message. A key is used to decode/decipher/discover the hidden message. This can be anything from a password, a pattern, a black-light, or even lemon juice.

In this paper I will focus on the use of Steganography within digital images (BMP and PNG) using LSB Substitution, although the properties of Image Steganography may be substituted with audio mp3's, zip archives, and any other digital document format relatively easily.

2. In-Depth Discussion

When first deciding what topic to research and study I was initially interested in encryption algorithms and ciphers. I have always been intrigued with security implementation and breaching, so I began developing an encryption algorithm to test its strength. Soon, with the help of some friends, I discovered Image Steganography and it only seemed natural that I take my encryption algorithm and

apply it to the newfound subject. Further research, development, and intrigue led to me to develop attacks on Image Steganography (attempting to discover if an image has Steganography applied to it).

2.1 Applications

Image Steganography has many applications, especially in today's modern, high-tech world. Privacy and anonymity is a concern for most people on the internet. Image Steganography allows for two parties to communicate secretly and covertly. It allows for some morally-conscious people to safely whistle blow on internal actions; it allows for copyright protection on digital files using the message as a digital watermark. One of the other main uses for Image Steganography is for the transportation of high-level or top-secret documents between international governments. While Image Steganography has many legitimate uses, it can also be quite nefarious. It can be used by hackers to send viruses and trojans to compromise machines, and also by terrorists and other organizations that rely on covert operations to communicate secretly and safely[2].

2.2 Implementation

There are currently three effective methods in applying Image Steganography: LSB Substitution, Blocking, and Palette Modification[1]. LSB (Least Significant Bit) Substitution is the process of modifying the least significant bit of the pixels of the carrier image. Blocking works by breaking up an image into "blocks" and using Discrete Cosine Transforms (DCT). Each block is broken into 64 DCT coefficients that approximate luminance and color—the values of which are modified for hiding messages. Palette Modification replaces the unused colors within an image's color palette with colors that represent the hidden message.[1]

I have chosen to implement LSB Substitution in my project because of its ubiquity among carrier formats and message types. With LSB Substitution I could easily change from Image Steganography to Audio Steganography and hide a zip archive instead of a text message. LSB Substitution lends itself to become a very powerful Steganographic method with few limitations.

LSB Substitution works by iterating through the pixels of an image and extracting the ARGB values. It then separates the color channels and gets the least significant bit. Meanwhile, it also iterates through the characters of the message setting the bit to its corresponding binary value[3].

```
int setLeastBit(int color, int value){
   return (((color >>> 1) << 1) | value); //LSB becomes: 0 or 1
}

int count = 0;
for(int j = 1; j < image.getHeight(); j++){
        for(int k = 0; k < image.getWidth(); k++){
        argb = image.getRGB(k, j); //get pixel value
        a = getA(argb);            //alpha
        r = getR(argb);            //red
        g = getG(argb);            //green
        b = getB(argb);            //blue

        r = setLeastBit(r, parseInt(binaryMessage.charAt(count ++)));
        if(count >= binaryMessage.length()){
           image.setRGB(k, j, getIntFromARGB(a, r, g, b));
           break;
        }
        …Green, Blue, etc.
     }
     image.setRGB(k, j, getIntFromARGB(a, r, g, b));
}
```

In my implementation, I first encrypt the message, and then I create a header that evaluates the corresponding mode, length, and offset to Steganograph the image with. By using a header I am able to have more control over the type of information I am hiding as well as the carrier file I am using.

I encrypt the message using a modified Vigenere Cipher that takes advantage of the entire ASCII character set, implements passwords, and uses simple hashing. A Vigenere Cipher is based on the Caesar Cipher which uses an alphabetic shift algorithm[5]. A Caesar Cipher shifts the alphabet resulting in:

xyzabc assigns: a=x, b=y, c=z, d=a, e=b, f=c

A Vigenere Cipher usually implements a table of 26 Caesar Ciphers (one for each character), allowing it to create a poly-alphabetic encrypted message (where one character can be represented by multiple different characters: ‘111’ -> ‘aKs’ based on the shift).

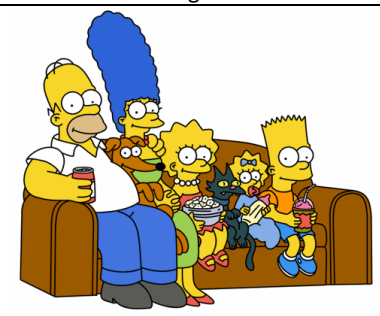
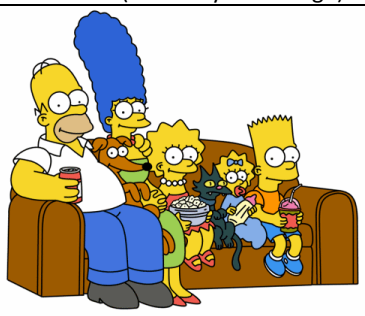
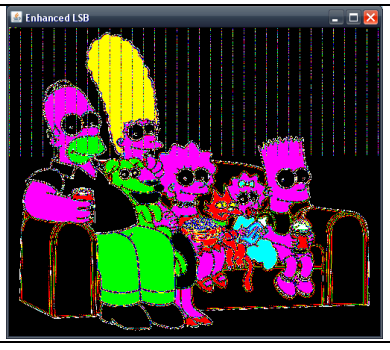
a: wxyzabc
b: xyzabcd
c: yzabcde
d: zabcdef
e: abcdefg

2.3 Detection/Attacks

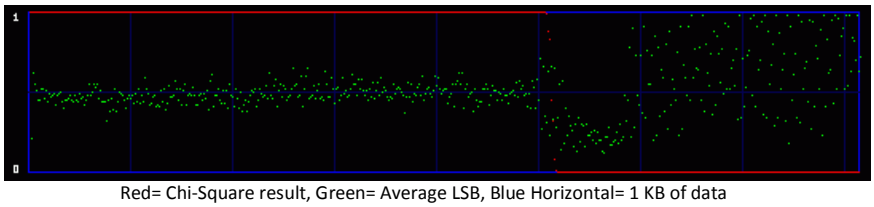
While the purpose of Steganography is to hide messages, it may not be very effective at doing so. There are several attacks that one may execute to test for Steganographed images—they are: Visual Attacks, Enhanced LSB Attacks, Chi-Square Analysis, and other statistical analyses. In performing a visual attack you must have the original "virgin" image to compare it the Steganographed image and visually compare the two for artifacts. In the Enhanced LSB Attack, you process the image for the least significant bits and if the LSB is equal to one, multiply it by 255 so that it becomes its maximum value.

```
int getEnhancedLSB(int argb){
   int a = getA(argb);
   int r = getLeastBit(getR(argb)) * 255;
   int g = getLeastBit(getG(argb)) * 255;
   int b = getLeastBit(getB(argb)) * 255;
   return  getIntFromARGB(a, r, g, b);
}

for(int j = 0; j < image.getHeight(); j++){
   for(int k = 0; k < image.getWidth(); k++){
      int argb = image.getRGB(k, j);
      argb = getEnhancedLSB(argb);
      image.setRGB(k, j, argb);
   }
}
```

Original	Modified (~2000 byte message)
	
	

Chi-Square Analysis calculates the average LSB and constructs a table of frequencies and Pair of Values; it takes the data from these two tables and performs a chi-square test. It measures the theoretical vs. calculated population difference[6]. The Chi-Square Analysis calculates the chi-square for every 128 bytes of the image. As it iterates through, the chi-square value it calculates becomes more and more accurate until too large of a dataset has been produced. Here is the output of a chi-square analysis on a Steganographed image:



Red= Chi-Square result, Green= Average LSB, Blue Horizontal= 1 KB of data

The red line, a value close to 1, indicates that there is Steganography within the first 5KB of the image. The graph also shows that the LSB's are quite similar during that range, which further supports the conclusion. Because this attack relies on statistical analysis it cannot detect patterns or Steganography on very complex images with lots of noise than one can detect through visualization of the Enhanced LSB's.

2.4 Benefits/Drawbacks

With LSB Substitution it is quite easy to tell if an image has been Steganographed with an Enhanced LSB Attack. A complex image yields a much better Steganographed image—in that is harder to visually detect Steganography. Chi-Square Analysis can detect Steganography much better than Enhanced LSB's; however, one can still construct an image to account for statistical irregularities so that when applying Steganography to an image, they can make sure to preserve (as best as possible) the statistical frequencies so that a chi-square analysis fails to produce a qualified result.

In my project I used a header to help with the embedding and extracting of a message from an image. This allows for quick updates to the encoding algorithm, as well as multiple user-specific modes of encoding. It also allows for better Stengagraphic hiding because the header format changes depending on the message type, mode, and length, making it harder to crack and detect my Steganography.

Encryption of the message helps to improve the security of the message. Because of the encryption's poly-alphabetic nature it makes it a lot harder to crack. However, if one can determine the length of the password, the following rule applies:

```
charAt(i) = charAt(length+i)
```

Once the length is found, the Caesar Cipher must be cracked by performing frequency analysis on the characters and one can (with time) determine the password. However, my implementation pre-encrypts the data so that even before a user applies a password, the message is already encrypted. This renders the frequency analysis useless because it implements the entire ASCII table, not just the alphabet, and it can have more than one character represent another. The complexity of frequency analysis increases quickly and largely on a poly-alphabetic message.

2.5 Future Thoughts

I hope to add support to hide all file formats. This allows for a much broader spectrum of uses: one would be able to encode .exe, .doc, .pdf, .mp3, etc. The program would be more versatile because often hiding text just isn't enough.

I also would like to implement batch image processing and statistical analysis so that I can run the program through a dataset of images and detect Steganography and perhaps crawl through Google Image Search to see how prevalent Steganography is.

I eventually plan to port the program to use C/C++ so that I may take advantage of bit-fields in C and learn to code GUI's as well. I have a plug-in handler developed for C++ that I would like to use in this project so that third-party developers may contribute to the project.

3. Conclusions

With this project I have learned a lot, especially about bit operations and bit-masking, something that I never understood before. This project was fun from the start and only got more interesting as I went on developing it. I became more interested in the subject the more I researched it.

I have learned that while implementing Image Steganography is important, thinking of how to detect and attack it and the methods to do so are far more complex than actually doing the Steganography itself. There is a lot of research that is beginning to discover new ways to detect Steganography, most of which involves some variation of statistical analysis. It is interesting to see what other methods will be developed and how accurate they will be at detecting Steganography.

References

1. Kesslet, Gary C. An Overview of Steganography for the Computer Forensics Examiner, Burlington, 2004.
2. Lin, Eugene and Edward Delp: A Review of Data Hiding In Digital Images, West Lafayette, 1999.
3. Hosmer, Chet. Discovering Hidden Evidence, Cortland, 2006.
4. Fridrich, J., R. Du, M. Long: Steganalysis Of LSB Encoding In Color Images, Binghamton, 2007.
5. Vehse, Heymo. YAVI: Yet Another Vigenere Algorithm www.leafraker.com/yavi
6. Guillermito. Steganography: A Few Tools to Discover Hidden Data, 2004. http://guillermito2.net/stegano/tools/index.html

Appendix

Encryption

```
String Encode_Once(String msg, String key)
{  //Credits: Heymo Vehse, www.leafraker.com/yavi
   String encrypted = "";
   for(int i = 0; i < msg.length(); i++)
   {
      if(Character.isISOControl(msg.charAt(i)))
      {
         encrypted += msg.charAt(i);
      }
      else
      {
         char ch = key.charAt(i % (key.length()));
         String curKey = String.valueOf(ch);
         int hash = curKey.hashCode() - 31;
         int n = msg.charAt(i) + hash;

         if(n > 126)
         {  //Force only ASCII characters
            n -= 95;
         }
         encrypted += (char)n;
      }
   }
   return encrypted;
}
```

The Header

```
public int theHeader(int pixels, bool encrypted, int mode, int length)
   {  //Header Format(bits)
      //encrypted(1): [1]
      //mode:(3):     [000]
      //step:(4):     [0000]
      //length(24):   [0000,0000,0000,0000,0000,0000]

      if(length == 0)
         return 0;

      int step;
      int bits = length * 8;
      int bits_per_pixel = 0;
      int header = 0; //[0]

      if(encrypted)
         header = 1; //[1]

      if((mode & RED) == RED)
         bits_per_pixel++;
      if((mode & GREEN) == GREEN)
         bits_per_pixel++;
      if((mode & BLUE) == BLUE)
         bits_per_pixel++;

      header |= mode;//[000][0]

      int diff = (bits_per_pixel * pixels) - bits;
      if(diff < 0){
         new JOptionPane().showMessageDialog(null,
             "Message is too long.\n"+
             "Fix: Try using a larger image or smaller message.",
             "Error",
             JOptionPane.ERROR_MESSAGE);
         return 0;
      }
      else if(diff == 0)
         step = 0;
      else{
         step = (int)(diff / bits);
         if(step > MAX_STEP)
            step = MAX_STEP;
      }

      header = ((header << 4) | step);
      int temp = header << 24;
      header = temp | bits;

      return header;
   }
```