Least Significant Bit Steganography

Jason West
Clemon University
Clemson SC
jgwest@clemson.edu

Chaitanya Mandru
Clemon University
Clemson SC
cmandru@clemson.edu

ABSTRACT

In our project we implemented a basic least significant bit steganography algorithm. Its primary purpose is educational however it can still be used to efficiently conceal a text file in a png. Our algorithm uses an API called FreeImage for pixel manipulation which will need to be installed before use. We found that our algorithm effectively hides our message while changing the image in such a way that it remains indistinguishable from the original by the human eye.

Keywords

Steganography; Cybersecurity; Linux; Least Significant; Data Hiding; Steganographic algorithm; FreeImage

ACM Reference Format:

Jason West and Chaitanya Mandru. 2021. Least Significant Bit Steganography. In CPSC 6240: System Administration and Security Fall 2021 Clemson, SC. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXX/XXXXXXX

1. INTRODUCTION

We decided to implement a Least Significant Bit Steganography (LSB) algorithm because of the increasing importance of steganography in the digital world, particularly in industry for copyright control.[5] We focused on creating a LSB algorithm that would hide messages (written in a .txt file) in an image (in an RGB or RGBA .png format). We chose this method because there are many advanced algorithms with varying degrees of success, however, this one is both simple and effective.[5] Additionally, we wanted to keep things simple with the intention for basic use (particularly academic). This way those who are unfamiliar with steganography can see its implementation and effects in a simple program before learning more advanced techniques that might be overwhelming for a beginner.

2. Background

Steganography is the study of hiding a message inside another object. The file to hide is referred to as the message, while the image or object it is hidden in is called the cover. A stego-object is the result of merging the message with the cover. Steganography is similar to cryptography in that it intends to keep some information private, however it differs from cryptography in that it does so primarily through obscurity (concealing the very existence of the information itself). However, steganography can be combined with cryptography to further protect sensitive information by compressing and encrypting the message before hiding it in the cover. Steganography has a variety of uses, from espionage to copyright protection.[4][5]

Steganography in copyright protection is used in two different ways, watermarking and fingerprinting. Watermarking is used to identify who owns the rights to the material. This is done by hiding some identifier or information in the copyright material. Whereas fingerprinting is used to identify who has violated their licensing/copyright agreement by sharing the material. This is done by hiding a unique identifier in the material. This way if a pirated version of the material appears it can be determined which individual can be held accountable for its distribution.[4][5]

Another use for steganography is for data verification and identifying evidence of tampering[1] in something such as an RFID for a manufactured good. Imbedding a destructible watermark in a way such that if the RFID is tampered with in some way will result in the destruction of the watermark. If the watermark is destroyed, then the owner of the RFID then knows their system has been compromised and anything pertaining to that RFID should be investigated and the good the RFID identified should be scrapped. This is particularly important for goods that are either important for national security (military defense systems, computer chips, etc.) or goods that could potentially result in catastrophic loss of life (airplane parts, steel beams for building, etc.).[1]

3. Motivations and Objectives

We sought out to implement a basic LSB algorithm to hide a message in a txt file in a cover png. The stego-object must be indistinguishable from the original by the human eye. Additionally, the ability to unhide a message from a stego-object would be required. This way two parties can secretly exchange messages using our algorithm where the sender uses the hide function and the recipient uses the unhide function.

4. Methodology and Design

4.1 PNG Pixels and ASCII Characters

Our algorithm focuses on png images; this is because they have lossless compression, thus guaranteeing if the stego-object is to be compressed and emailed to the recipient that the message will not be destroyed. This also means that if our algorithm is used as a form of watermarking or fingerprinting then the identifier inside the image would not be destroyed or removed when pirated.

Pixels in a png are represented most often in RGB (24 bits, Red, Green, Blue) and RGBA (32 bits, RGB with Alpha value for

[1] This is something that has been discussed in my graduate research meetings at a cyber security research institute

CPSC 6240, Fall 2021, Clemson, SC

ACM ISBN XXX-X-XXXX-XXXX-X/XX/XX.

transparency) with 8 bits per value. Using FreeImage we are able to guarantee RGBA by converting any non RGBA representation into one or terminating the program and warning the user if not. Because we can guarantee 32 bits per pixel when hiding we chose to hide 2 bits per RGBA value; this lets us hide one character (which is represented by an 8 bit ASCII value) per pixel while still being able to maintain indistinguishability to the human eye. Being able to hide 1 character per pixel also lets us double the amount of information we can hide in an image compared to 1 bit per RGBA value thus allowing for a smaller cover object for any given message.

Prior to message insertion into the cover the size of the files are compared to ensure the message can be hidden. Because of the 1:1 nature of the message size in bytes and image pixels there is a simple check of:

if message_size + 1 > image_width*image_height then return error

The +1 comes from the addition of a terminating sequence to the message so that when extracting the message from the stego-object we will know if the entire message has been retrieved. The value for "end of text" in ASCII was chosen as the terminating sequence.

4.2 Hiding Bits

Say we have an orange pixel as shown in Figure 1 with rgba (240, 125, 17, 255) and the letter "H" (0100 1000 in ASCII) that we wanted to hide the first 2 bits inside the last 2 bits of r. Then to hide H we would:

Figure 1: Orange Pixel

H = 0100 1000 = 72

r = 1111 0000 = 240

1. Extract relevant bits (underlined) from H with relevant bitmask
    a. H & bitmask = result1
        i. 0100 1000 & 1100 0000 = 0100 0000
2. Shift extracted bits to be in least significant positions
    a. result1 >> 6 = result2
        i. 0100 0000 >>6 = 0000 0001
3. Replace last two bits of r with result2
    a. r | result2 = newR
        i. 1111 0000 | 0000 0001 = 1111 0001 = 241

Note that in the end that r could only change by at most a value of 3. This is critical because small changes in these values are imperceptible to the human eye.

This process is repeated for each RGBA value and the entirety of the message. We chose to hide first two bits of a character in r, the next two in G and so on. Characters are hidden starting from the bottom left pixel moving to the top right. Once the entire message has finished being hidden the terminating sequence is inserted into the image. If there are remaining pixels that do not need to be changed then their original values are simply copied over to the stego-object.

4.3 Unhiding Bits

Say we now have a stego-object that has a message hidden in it and we would like to extract our message then the process is similar to hiding them. Recall our stego-object pixel from before:

Stego-object r = 1111 0001

1. Extract relevant bits using a bitmask
    a. r & bitmask = result1
        i. 1111 0001 & 0000 0011 = 0000 0001
2. Shift bits to their original position in the ASCII character
    a. result1 << 6 = result2
        i. 0000 0001 << 6 = 0100 0000
3. Combine all values extracted from the stego-pixel's RGBA values
    a. result2_r & result2_g & result2_b & result2_a = H
        i. 0100 0000 & 0000 0000 & 0000 1000 & 0000 = 0100 1000 = H
    b. (note our example did not show what the values for g, b, and a would be but the values shown are correct)

This process is repeated for all pixels starting at the bottom left and moving towards the top right until the terminating sequence is extracted. If the terminating sequence is found the extracted message is reported. Otherwise, there was no message in the image that used our algorithm to imbed it.

5. Analysis and Results

Our algorithm works as intended and successfully hides the message in the image and extracts the message when unhiding. We first verified the hiding function by hand and ensured the pixels were changing to the appropriate values in the stego-object. Verifying unhide was much easier given we could easily check the output being the correct message that we originally inserted. Additionally, we verified that we could not insert a message that was too large by using files of known sizes and checking the output. Visual inspection can show that the original and stego-objects are visually identical as shown by figure 2.

Figure 2 Stego-Object with H and terminating sequence inserted

6. Conclusion

While our algorithm might not be secure to steganalysis it is sufficient for the purpose with which we have defined it. It is ideal for a basic introduction to steganography for those who are interested in learning more. It's also just a fun way to exchange messages with friends.

7. Future Works

Firstly, with more time we would like to add support for other image formats, audio, and video as cover objects as well as various steganographic algorithms. Secondly, we would also like to add a key for input that will act as a seed for a pseudo random number generator that can be used for pixel selection when imbedding the message into the cover. Third, we would like to be able to add flags to compress and encrypt the message before being hidden in the cover. Fourth, we would like to add support for other forms of steganography such as image to image. Finally, we would like to verify that our algorithm(s) would work for all message file formats that use an ASCII representation.

8. User Manual

8.1 How to Obtain:

Download the files

8.2 How to Use

To run successfully on your systems, you need to install all the dependencies. Please run the following commands to install all the necessary dependencies in your system.

sudo apt-get update

sudo apt-get install -y libfreeimage-dev

8.2.1 How to Hide Message

Once all the dependencies are successfully installed. You need to compile the program and run it. Open a terminal window and move to the directory where you have downloaded the project.

command to compile: make hide

Command for running: ./hide message.txt input.png output.png

Where message.txt is the name of the input file containing the information (which you want to hide)

input.png is the name of the input png file name in which you want to hide the message in.

output.png is the name that you wanted to identify the created png file containing the hidden message.

After running the program you can see that a new file .png file is created with the name output.png which contains the information.

8.2.2 How to Unhide Message

Command to compile: make unhide

Command to run: ./unhide input.png message.txt

Where input.png is the name of the file that you wanted to extract information from.

message.txt is the name that you have given to the output file which contains the extracted information.

After running the program you can see a message.txt file is created in that directory that contains the message extracted from the png file.

9. REFERENCES

[1] Chin-Chen Chang and Yung-Chen Chou. 2009. A fragile digital image authentication scheme inspired by WET paper codes. Fundamenta Informaticae 90, 1-2 (2009), 17–26. DOI:http://dx.doi.org/10.3233/fi-2009-0002

[2] Msallam, Mohammed. (2020). A Development of Least Significant Bit Steganography Technique. 20. 31-39. 10.33103/uot.ijccce.20.1.4.

[3] Reddy, V. Lokeswara, A. Subramanyam, and P. Chenna Reddy. "Implementation of LSB steganography and its evaluation for various file formats." Int. J. Advanced Networking and Applications 2.05 (2011): 868-872.

[4] Singh, Arun Kumar, Juhi Singh, and Harsh Vikram Singh. "Steganography in images using lsb technique." International Journal of Latest Trends in Engineering and Technology (IJLTET) 5.1 (2015): 426-430.

[5] Tyagi, Vikas. "Image steganography using least significant bit with cryptography." Journal of global research in computer science 3.3 (2012): 53-55.

[6] Umamaheswari, M., S. Sivasubramanian, and S. Pandiarajan. "Analysis of different steganographic algorithms for secured data hiding." IJCSNS International Journal of Computer Science and Network Security 10.8 (2010): 154-160.