Digital Steganography: A Symmetric Key Algorithm

By: Joshua C. Clark
Background/History

- Steganos- “covered/protected”
- Graphy- “Writing”
- Obscurity in “plain sight”
  - Differs from traditional cryptography
- Physical Applications from Ancient Greece
  - Wax Tablets
  - Tattooing heads of slaves
- Evolution and adaption in WWII
  - Invisible Inks (citrus based ink)
Modern Applications: Digital Steganography

- Concealing data in images or audio files. (Effective for any digitally transmitted packets)
- Rumors of al Qaeda use for hiding messages in online photographs
- Russian spies imbedding sensitive information in ordinary pictures
- Digital Watermarking
According to the FBI, this photo has an imbedded image of an airport.
Image of Professor Joyner from his website.

Example of Digital Watermarking. The image on the left was opened with the program Frhed, a hexadecimal editor tool.
Pictures are a combination of individual pixels.

Each pixel represents a color intensity value for the primary color bands (Red, Green, and Blue).

- Example: Orange = 100% R + 50% G + 0% B

This pixel arrangement can be thought of as a matrix.
Mathematics and Image Processing Continued

The image matrix has dimensions: MxNx3

- M: Number of pixel rows
- N: Number of pixel columns
- 3: Due to the three color spectrums (RGB)
Computer Storage

- Binary (1’s or 0’s) and Hexadecimal (0-F)
  - 1 byte = 8 bits=2 Hex= 1 ASCII Characters
  - Example:
    - 100% **R**: 1111 1111 = FF
    - 50% **G**: 0111 1111 = 7F
    - 0% **B**: 0000 0000 = 00
    - **Orange**: FF7F00
### ASCII Message “HI”:

**Hexadecimal:** 48 49  
**Binary:** 0100 1000 0100 1001
Existing Freeware and Algorithms: EasyBMP

- Let \( N \) (a byte) be a number from 0 to 255
- Let \( r_1, g_1, b_1, a_1, r_2, g_2, b_2, a_2 \in \{0,1\} \)
- \( N = r_1 + g_1 \cdot 2 + b_1 \cdot 2^2 + a_1 \cdot 2^3 + r_2 \cdot 2^4 + g_2 \cdot 2^5 + b_2 \cdot 2^6 + a_2 \cdot 2^7 \)
- If \((R_1,G_1,B_1,A_1)\) and \((R_2,G_2,B_2,A_2)\) are adjacent pixels to be overwritten, then the new pixels are obtained by:
  - \((R_1,G_1,B_1,A_1) - \lfloor (R_1,G_1,B_1,A_1) \rfloor + (r_1,b_1,g_1,a_1)\)
  - \((R_2,G_2,B_2,A_2) - \lfloor (R_2,G_2,B_2,A_2) \rfloor + (r_2,b_2,g_2,a_2)\)
Use pixels (255,255,255,0) and (255,255,255,0) and a message
EasyBMP Algorithm (Finale):

- Therefore, the new pixels are:
  - $(255, 255, 255, 0) - (255, 255, 255, 0) \% 2 + (0, 1, 0, 0)$
    $= (255, 255, 255, 0) - (1, 1, 1, 0) + (0, 1, 0, 0)$
    $= (254, 255, 254, 0)$

- $(255, 255, 255, 0) - (255, 255, 255, 0) \% 2 + (1, 0, 0, 0)$
  $= (255, 255, 255, 0) - (1, 1, 1, 0) + (1, 0, 0, 0)$
  $= (255, 254, 254, 0)$
THIS IS A TEST TO SEE IF I CAN DETECT AN IMAGE CHANGE USING PAINT PROGRAM AND BUCKET FILL FEATURE.

 Parameter 1  Parameter 2  Parameter 3

 Command Prompt

```
C:\Users\n141152\Desktop\SM4738RYPTO\Steganography>Stego
EmbedText usage:
This embeds a text file into a graphic file, and saves the result to a new file (-e for encode):

   Stego -o <input file> <input bitmap> <output bitmap>
This extracts a hidden text message and outputs the result
(-d for decode):

   Stego -d <input bitmap>
This gives this help information:

   Stego -h

Created on February 3, 2006 by Paul Macklin.
Uses the EasyBMP library, Version 0.71.
Licensed under GPL v. 2 by the EasyBMP Project.
Contact: http://easybmp.sourceforge.net
```

G:\Users\n141152\Desktop\SM4738RYPTO\Steganography>Stego -o LongMessage.txt white.bmp white.bmp

whiteoutput.bmp  Date modified: 3/5/2013 10:00 PM  Size: 764 KB
Bitmap image  Dimensions: 450 x 446  Date created: 3/5/2013 9:59 PM
EasyBMP is easily susceptible to Paint’s auto fill command. This is because the encryption starts in the upper left corner and runs from left to right and top to bottom.
Symmetric Key Algorithm by Least Significant Bit (LSB) Imbedding

- Example:
  - Message: “F” = 46 Hex = 0100 0110 Binary

- Original Pixels:
  - 10110001 01110010
  - 11101110 10110110
  - 11111111 11001100
  - 00000000 01100111

- Output Pixels:
  - 10110000 01110011
  - 11101110 10110110
  - 11111110 11001101
Encryption Algorithm

- Read an image into MATLAB
- Input message to be encrypted
  - Convert the ASCII message into Binary for LSB embedding
- Choose a two component key (x,y):
  - x = “set seed”
  - y = “wasted” random numbers
- Separate the image matrix into R, G, and B matrices
- Imbed message in the LSB of the randomly selected pixels across the RGB spectrum
Decryption Algorithm:

- Upload the encrypted image
- Input the (x,y) key
- Allow the code to extract the “randomly” hidden message from the corresponding LSB’s.
Security Improvements

- Assuming a 40 ASCII character encrypted image.
  - Also assume the location, though not the encryption order, of the altered LSB’s are known.
- Brute Force Attempted Crack = (40*8)!
  - A LARGE key space, safe from a brute force attack.
Questions?

Can you spot the message?

Input Message: ‘SECRET MESSAGE!’

Output MESSAGE =

SECRET MESSAGE!