*In The Name Of God,*
*The compassionate, The Merciful*

# Steganography

## The System Programming Project
## First Report

*Dr. Ali Hamze*

*Negar Foroutan Eghlidi, Parisa Akaber,*
*Mahyar Hosseini Motlagh,*
*Rojin Babayan, Arash Pourhabibi Zarandi*

*Negar Foroutan Eghlidi, Parisa Akaber,*
*Mahyar Hosseini Motlagh,*
*Rojin Babayan, Arash Pourhabibi Zarandi*

# Steganography :

Steganography is defined as “the art and science of communicating in a way which hides the existence of the communication” . Methods of steganography have existed for centuries, though with the advent of digital technology, have taken on a new form. Embedding data within the redundancy and noise of media files is among these digital techniques. In this project we will implement the process involved with hiding messages and files within uncompressed bitmap image files. We will use the least significant bit (LSB) process.

Examples of steganography date as far back as 440 B.C., where Histiæus was said to shave the heads of slaves and tattoo messages on them. Once the hair had grown back, the message was effectively hidden until the receiver shaved the heads once again. Another technique was to conceal messages within a wax tablet, by removing the wax and placing the message on the wood underneath.

Steganography in digital technology is often associated with files which require a human perspective to verify the integrity and quality. This includes the realm of media files (video, audio, images). Image files lend themselves to exploitation particularly well, which we will implement throughout this project and we’ll focus on the bitmap formatted images (BMP).

## Steganography Terms

- **Carrier File** – A file which has hidden information inside of it.
- **Steganalysis** - The process of detecting hidden information inside of a file.
- **Stego-Medium** - The medium in which the information is hidden.
- **Redundant Bits** - Pieces of information inside a file which can be overwritten or altered with out damaging the file.
- **Payload** - The information which is the be concealed.

# Bitmap Image (BMP) :

Bitmap images are were introduced by Microsoft to be a standard image file format between users of their Windows operating system. The file format is now supported across multiple file systems and operating systems, but is being used less and less often. A key reason for this is the large file size,resulting from poor compression and verbose file format. This is, however, an advantage for hiding data without raising suspicion. To understand how bitmap images can be used to conceal data, the file format must first be explained.

## The bitmap file format :

A bitmap file has two main parts, the header and the data.The header file, which consitsts of 54 bytes, has also two sub-parts, the Bitmap Header and the Bitmap Information, however image which are less than 16-bit have another sub-part within the header called Color Palette. The Bitmap Data is followed by the header part.

### Bitmap Header :

It is refered to the identification of a valid bitmap image, this is done majorly by the first two bytes, which are 0x42 and 0x4D (in ASCII : BM). After that it has 4 bytes to express the size of the file, the following 8 bytes are reserved for application identification and offset lenghts. So we have :

- 0-1 Bitmap identifier 0x42 0x4D
- 2-5 Bitmap file size
- 6-9 Reserved
- 10-13 Bitmap data offset

Here is the following bytes which consists of the Bitmap Information part which is located in 30 bytes starting at byte fourteen :

- 14-17 Bitmap header size
- 18-21 Bitmap image width
- 22-25 Bitmap image height
- 26-27 Number of color planes
- 28-29 Bitmap color depth
- 30-33 Bitmap compression method
- 34-37 Bitmap data size

- 38-41 Bitmap image horizontal rule
- 42-45 Bitmap image vertical resolution
- 46-49 Number of colors used
- 50-53 Number of important colors used

## Bitmap Data :

This part of bitmap file contains the actual image, stored as pixels. The image stores itsef in reverse, it means the first line of data corresponds to the bottom line of an image, moving its way up, and also, the RGB color format is stored reversed too, which means Bule comes first, followed by Green and then Red. The property that is important for us who want to impliment steganography on this image format is the fact that the pixels are explicity written out in the file which allows identification or modification easily.

Here is a Small part of the Bitmap Data block of an bitmap image :

```
35DAF237 D7F247DC F246D6F2 47D6EF4F 5..7..G..F..G..O
DBF352DA EF4EDCF3 44DCF44E D9F245DE ..R..N..D..N..E.
F548DEF3 52DBF04D DAF249D8 EF51D6EE .H..R..M..I..Q..
4BCFEB47 DCF34DDB F24AD8F3 4EDBF346 K..G..M..J..N..F
D9EF41D8 EF43D3F0 46D6ED49 D6EF47DB ..A..C..F..I..G.
F14BDAF2 45D1EE47 D3EC45D5 EF47CEED .K..E..G..E..G..
42D3ED3B D3F146D8 F247D5F0 3ED5F03A B..;..F..G..>..:
D9F145DA F43DD1EE 3BD2EE3E D3EF39D1 ..E..=..;..>..9.
EC35CFEB 3DD6EF3F D2EE36CD EA32D4EE .5..=..?..6..2..
3CCFEE42 D3F13DCD EB3DCBEA 39CEED3F <..B..=..=..9..?
D3EE41CD EB34D1EF 39D0EB3C CEEB34CF ..A..4..9..<..4.
EB35D4EE 38D1EC36 D4EE41D3 EE31C9E8 .5..8..6..A..1..
35D2F03B D1EF3BCE EC33CBE9 35CDEC39 5..;..;..3..5..9
CFEE34D1 ED33CDEC 3BCFEE38 D1EF33CA ..4..3..;..8..3.
EA38CCEA 3AD1EE40 D2F036D0 ED36D2ED .8..:..@..6..6..
35CBEA3B CEEC34CB EB30CCEA 33D2EF34 5..;..4..0..3..4
D2EF32CC EA39CCEB 3BCBED37 CEEE3AD0 ..2..9..;..7..:.
EC34CCEB 30CCEB32 CFEE31D2 EE35D2EF .4..0..2..1..5..
32CBEB35 CDEB37D1 EE2FCBEB 34CDEC39 2..5..7../..4..9
CAEB32D1 EF31D1ED 33CEEC36 CFEC39D2 ..2..1..3..6..9.
EF37C7EA 33CDEB31 D2ED32CE EC38D1EE .7..3..1..2..8..
```

For an uncompressed 24-bit bitmap image, each pixel is represented by 24 bits, or three bytes. Each byte is refered to each of three colors, Red, Green or Blue. For example a black pixel is reperesnted in binary by 00000000 00000000 00000000. As we have 8 bits, we have 2^8 = 256 possibilities or different colors for a plane and 256^3 = 16777216 possibilities for the color of a pixel.

# Our Method:

The method we'll use to implement our Steganography project is called LSB or Least Significant Bit. However, it might be different from the original LSB method and we might improve it. Let me illustrate the method with an example :
First we have to break the text into words and letters actually, then each character must be converted to it's ASCII code. For example, consider the character 'G' that we want to embed it in the picture. The program reads the image data and changes data into the binary number system. It chooses some pixels for embedding the text. Consider 8 bytes of the carrier are like this (the least significant bits are underlined):

| | | | |
|---|---|---|---|
| **1001010<u>1</u>** | **0000110<u>1</u>** | **1100100<u>1</u>** | **1001011<u>0</u>** |
| **0000111<u>1</u>** | **1100101<u>1</u>** | **1001111<u>1</u>** | **0001000<u>0</u>** |

Character 'G' is represented in the American Standard Code for Information Interchange (ASCII) as 71 and as a binary string is 01000111. Here we change the least significant bits of the 8 bytes and replace them with the digits of our binary string "01000111" :

| | | | |
|---|---|---|---|
| **1001010*<u>0</u>*** | **0000110*<u>1</u>*** | **1100100*<u>0</u>*** | **1001011*<u>0</u>*** |
| **0000111*<u>0</u>*** | **1100101*<u>1</u>*** | **1001111*<u>1</u>*** | **0001000*<u>1</u>*** |

In the sample above, only half of the least significant bits were actually changed (shown above in italics). This makes some sense when one set of zeros and ones are being substituted with another set of zeros and ones. We can do this job by using a bitwise operation in C &. In this example we actually do the 1-bit LSB, which means we used

just the least significant bit of a byte, however we can use the two least significant bits of a byte (2-bit LSB) and so on.

So by this way we can embed each character in the image. Now we might use an optional method to make our Steganography better, which means it can harder been found. We use the "Pixel Value Differencing" or PVD to determine how many secret bits should be embedded. By this we understand that usually the edge areas can tolerate more changes than the smooth areas. So, our method will be adaptive with the features of high capacity and low distortion.

This embedding strategy is based on the concept that edge areas can tolerate a larger number of changes than smooth areas. So we use PVD to distinguish between edge areas and smooth areas. We can divide the range [0,255] of difference values into different levels, for instance, lower level, middle level and higher level. For each consecutive pixels, both pixels are embedded by the k-bit LSB, which the value of k is decided by the level of the differences. Consider that if we have a high level difference, so our k has a large value. Here we use modified LSB substitution method. The concept of this method is to change the value of the Most Significant Bit (MSB) for reducing the square error between the original and embedded pixel. We must have this idea in our mind that the differences before and after embedding must belong to the same level.

The difference value *d* is computed for each nonoverlapping block with two consecutive pixels. The way of partitioning the image into two-pixel blocks runs through all of the rows in a raster scan. Now we use our l-m-h division (lower-middle-higher) to determine the *k*. So we have *l*-bit LSB for the lower division, *m*-bit LSB for the middle division and *h*-bit LSB for the higher division. Here is the table of the ranges :

| Lower-level | Middle-level | Higher-level |
|---|---|---|
| $R_1=[0,15]$ | $R_2=[16,31]$ | $R_3=[32,255]$ |

We can compute l, m and h by this way : $l \leq \log_2 |R_1|$ , $m \leq \log_2 |R_1|$ , $h \leq \log_2 |R_3|$. So in this example we have 3-4-5 division as l=3, m=4 and h=5. Here are the detailed embedding steps :

- Step 1: Calculate the difference value $d_i$ for each block with two consecutive pixels, $d_i = |p_i - p_{i+1}|$
- Step 2: From l-m-h division we'll find the correct *k* for the *k*-bit LSB.
- Step 3: By the *k*-bit LSB substitution method, embed k secret bits into $p_i$ and k secret bits into $p_{i+1}$ . So we now have $p_i'$ and $p'_{i+1}$ .
- Step 4: Apply the modified LSB substitution method to $p_i'$ and $p'_{i+1}$ .
- Step 5: Calculate the new $d_i'$ by $d_i' = | p_i' - p'_{i+1} |$.
- Step 6: If $d_i$ and $d_i'$ belongs to different levels, we'll readjust as follows :

  I. Case 6.1: If $d_i$ is in lower-level range, but $d_i'$ isn't. If $p_i' \geq p_{i+1}'$ readjust $(p_i', p_{i+1}')$ to better choice between $(p_i', p_{i+1}' + 2^k)$ and $(p_i' - 2^k, p_{i+1}')$; otherwise, readjust $(p_i', p_{i+1}')$ to be better choice between $(p_i', p_{i+1}' - 2^k)$ and $(p_i' + 2^k, p_{i+1}')$.
  II. Case 6.2 : If $d_i$ is in middle-level range, but $d_i'$ is in lower-level. If $p_i' \geq p_{i+1}'$ readjust $(p_i', p_{i+1}')$ to better choice between $(p_i', p_{i+1}' - 2^k)$ and $(p_i' + 2^k, p_{i+1}')$; otherwise, readjust $(p_i', p_{i+1}')$ to be better choice between $(p_i', p_{i+1}' + 2^k)$ and $(p_i' - 2^k, p_{i+1}')$.
  III. Case 6.3: If $d_i$ is in middle-level range, but $d_i'$ is in high-level. If $p_i' \geq p_{i+1}'$ readjust $(p_i', p_{i+1}')$ to better choice between $(p_i', p_{i+1}' + 2^k)$ and $(p_i' - 2^k, p_{i+1}')$; otherwise, readjust $(p_i', p_{i+1}')$ to be better choice between $(p_i', p_{i+1}' - 2^k)$ and $(p_i' + 2^k, p_{i+1}')$.
  IV. Case 6.4: If $d_i$ is in higher-level range, but $d_i'$ isn't. If $p_i' \geq p_{i+1}'$ readjust $(p_i', p_{i+1}')$ to better choice between $(p_i', p_{i+1}' - 2^k)$ and $(p_i' + 2^k, p_{i+1}')$; otherwise, readjust $(p_i', p_{i+1}')$ to be better choice between $(p_i', p_{i+1}' + 2^k)$ and $(p_i' - 2^k, p_{i+1}')$.

*** In step 6, the better choice, say $(x_i , x_{i+1})$, means that it satisfies the conditions that $| x_i - x_{i+1} |$ and $d_i$ belong to the same level and $x_i , x_{i+1}$ are in range [0,255], also the value of $(x_i - p_i)^2 + (x_{i+1} - p_{i+1})^2$ is smaller.

For example, consider the 3-4-5 division with $p_i = 64$ , $p_{i+1} = 47$, and secret data $b=10100000_{(2)}$. Since $d_i = 17$ (middle level) the pixels are embedded by 4-bit LSB substitution and have the result $p'_l = 72$ , $p_{i+1}' = 32$. So we apply the modified LSB sunstitution and we have now $p'_l = 58$ , $p_{i+1}' = 48$ and $d'_l = 10$ (lower-level). Now we have Case 6.2 for readjusting, then we use $(p_i', p_{i+1}') = (58+2^4 , 48) = (74,48)$. Hence, $d'_l = 26$ belongs to middle-level.