Algorithms of Information Security: Steganography

Faculty of Information Technology
Czech Technical University in Prague

December 8, 2022

Image Files

- Digital images can be displayed as twodimensional matrices with rows and columns containing the position (x, y) of the image, while the element states the color value in that position.
- Each image file in digital form is composed of pixels. Every pixel is similar to a very small point in picture.
- A pixel is the basic unit of bitmap graphics. This is a point with a defined color. Each pixel has a defined color, usually composed on the basis of RGB - Red Green Blue, i.e. the three basic colors from which all other colors are composed. Images are then created from pixels of different colors.
- Hiding data into the image files is another type of steganography. This method is the most common steganographic technique. There are several methods for hiding data into images.

LSB (Least Significant Bit) method

- The simplest method for hiding in images is the LSB method, which uses the least significant bits, in which the secret information is hidden in the least significant parts of the image.
- The idea is based on the assumption that a small change in the shade of a pixel's color is undetectable to the human eye, especially if more or less random pixels in the image are slightly changed.
- This technique changes the last bits in a byte to encode a message in an image where the red, green, and blue values of each pixel are represented by eight bits (one byte) ranging from 0 to 255 in decimal, or 00000000 to 11111111 in binary.

LSB (Least Significant Bit) method

- Changing the last two bits in the fully red pixel from 11111111 to 11111101 changes only the red value from 255 to 253, which creates an almost imperceptible color change to the human eye, but still allows us to encode data within the image.
- Hiding with LSB also has its disadvantages. With some graphic formats such as e.g.: GIF, it can happen that changing the last bit can mean a color change. Therefore, a change based on color similarity is needed. Swap an entire byte and not just one bit. In this way, we achieve colors very similar to the original color, or we can use a grayscale image.

Metoda LSB (Least Significant Bit)

Exercise.
Suppose we have the following 4 bytes

$$(10001001\ 11011010\ 10101010\ 00001011)$$

and we want to insert the message - 1010 into them using the LSB method.

Solution.

- We already have the message in binary form and it is 1010.
- Next, we insert this information into bytes using the LSB method and get the following result:

$$(10001001\ 11011010\ 10101011\ 00001010)$$

- The bits that make up the secret message are in bold and the bits that have changed from the original are underlined.

Exercise.
Suppose we have the following grid for 3 pixels of a 24-bit image:

$$(00101101\ 00011100\ 11011100)$$

$$(10100110\ 11000100\ 00001100)$$

$$(11010010\ 10101101\ 01100011)$$

and we want to insert a message into it using the LSB method: 200.

Solution.

- First we convert 200 to binary. So we have 11001000.
- Next, we insert the number 200 using the LSB method using the last bit into the pixel grid.
- We get the following result:

$$(00101101\ 00011101\ 11011100)$$

$$(10100110\ 11000101\ 00001100)$$

$$(11010010\ 10101100\ 01100011)$$

- The bits that make up the secret message are in bold, and the bits that have changed from the original are underlined.
- From the example, we can see that it is quite likely that the bits we want to write already match the ones written, and therefore there is no change.

Exercise.
Suppose we have the following 2 pixels of a 24-bit image:
The first is binary $(00110111\ 01010101\ 01101111)$ and the second is $(00010011\ 00111010\ 01011011)$ and we want to insert the letter A into them using the LSB method using the last two bits.

Solution.

- First, we convert the letter A into binary format. We know that the character A is decimal ASCII 65.
- So in binary we have 01000001. Next, we mark the replaced bits in bold and the last 2 bytes are intact, since we do not write more than the letter A:

$$(00110\mathbf{11}\ 01010\mathbf{01}\ 01101\mathbf{11})$$

$$(00010\mathbf{11}\ 00111010\ 01011011)$$

- We get the following result:

$$(00110\mathbf{0}1\ 01010\mathbf{0}0\ 01101\mathbf{00})$$

$$(00010\mathbf{0}1\ 00111010\ 01011011)$$

- Bits that have changed from the original are marked in bold.

Exercise.
Suppose we have the following grid for 3 pixels of a 24-bit image:

$$(00100111\ 11101001\ 11001000)$$

$$(00100111\ 11001000\ 11101001)$$

$$(11001000\ 00100111\ 11101001)$$

and we want to insert the letter C into them using the LSB method.

Solution.

- First, we convert the letter C into binary format. We know that the character C is decimal ASCII 67.
- So in binary we have 01000011.
- Next, we insert the letter C using the LSB method using the last bit into the pixel grid.
- We get the following result:

$$(00100110 \; 11101001 \; 11001000)$$

$$(00100110 \; 11001000 \; 11101000)$$

$$(11001001 \; 00100111 \; 11101001)$$

- The bits that make up the secret message are in bold, and the bits that have changed from the original are underlined.

PVD (Pixel Value Differencing) method

- PVD method is proposed by Wu and Tsai, based on embedding secret data into two consecutive pixels.
- Given a gray-value cover image F sized $M \times N$.
- The cover image F is partitioned into two-pixel blocks by runs through all the rows of cover image in a raster scan order such that $F = \{F_i \mid i = 1, 2, \ldots, \frac{M \times N}{2}\}$.
- By definition each F_i contains two consecutive pixels, say $P_{(i,L)}$ and $P_{(i,R)}$. Assume that the gray values of $P_{(i,L)}$ and $P_{(i,R)}$ are $P_{(i,x)}$ and $P_{(i,y)}$ respectively. The difference value of $P_{(i,x)}$ and $P_{(i,y)}$ can derived by

$$d_i = P_{(i,y)} - P_{(i,x)}.$$

PVD (Pixel Value Differencing) method

- Wu and Tsai also design a range table R which consists of n contiguous subranges. In other words, $R = \{R_j \mid i = 1, 2, \ldots, n\}$.
- The purpose of the range table is to provide information about the hiding capacity of each F_i. The lower and upper bound values of R_j are denoted by l_j and u_j respectively, where l_1 is 0 and u_n is 255. The width w_j of each R_j is selected to be a power of 2, and can be computed by $w_j = u_j - l_j + 1$.
- For example, the range widths of 8, 8, 16, 32, 64, and 128, which partition the total range of $[0, 255]$ into $[0, 7], [8, 15], [16, 31], \ldots, [128, 255]$.

PVD (Pixel Value Differencing) method

- The hiding capacity of two consecutive pixels can be obtained

$$b_i = \lfloor \log_2(w_j) \rfloor$$

. Here, b_i is the number of secret bits that can be hidden in F_i.

- Read b_i bits from the binary secret data stream and transform b_i into its decimal value b'_i.
- A new difference value d'_i can be generated by

$$d'_i = \begin{cases} b'_i + l_j & \text{for } d_i \geq 0 \\ -(b'_i + l_j) & \text{for } d_i < 0. \end{cases}$$

- Count $m = d'_i - d_i$.
- Now the secret data can be embedded into F_i by modifying its $P_{(i,x)}$ and $P_{(i,y)}$.

PVD (Pixel Value Differencing) method

- The details of the embedding criteria are as follows:

$$(P'_{(i,x)}, P'_{(i,y)}) = \begin{cases} (P_{(i,x)} - \lceil \frac{m}{2} \rceil, P_{(i,y)} + \lfloor \frac{m}{2} \rfloor) \\ \text{if } m \text{ is odd} \\ (P_{(i,x)} - \lfloor \frac{m}{2} \rfloor, P_{(i,y)} + \lceil \frac{m}{2} \rceil) \\ \text{if } m \text{ is even} \end{cases}$$

Note: The sign $\lceil\ \rceil$ is rounding up, for example $4.3 = 5$ and $-4.8 = -4$.
The sign $\lfloor\ \rfloor$ is rounding down, for example $4.3 = 4$ and $-4.8 = -5$.

- Replace $P_{(i,x)}$ and $P_{(i,y)}$ in the cover image with the new pixel values so that the embedding process is accomplished.

OpenStego - https://www.openstego.com/

- OpenStego is a steganography application that provides two functions:
  - Data Hiding: Can hide any data in the cover file (for example: images).
  - Watermark: Watermark files (for example: images) with an invisible signature. It can be used to detect unauthorized file copying.
- This tool is written in Java and should run on all platforms supported by Java. MS Windows and Linux are supported and it shouldn't have a problem on other platforms either.
- It allows to encrypt information based on AES 128 and AES 256 algorithms.
- It uses the RandomLSB (Randomized LSB) algorithm to hide information and the Dugad algorithm to create the watermark.
- Examples of supported input files are JPEG, GIF, BMP or PNG.

Exercise.

First, install the OpenStego software.

- Create a secret message (.txt file) that you want to hide in the image. Choose an image to hide the message in. Use the OpenStego software to hide some secret text in your chosen image and save the resulting file.
- Use OpenStego software to get the secret message stored in the new image.

We can also create a watermark to verify images with our signature. You need to generate a signature file first and then it can be used to create a watermark or verify it later.

Exercise.
Using OpenStego software:

1. Generate a signature file that you then use to create a watermark.
2. Create a watermark.
3. Verify the watermark.

Mathematica provides built-in support for both programmatic and interactive image processing, fully integrated with Mathematica's powerful mathematical and algorithmic capabilities. You can create and import images, manipulate them using built-in functions, apply linear and non-linear filters and visualize them in many ways. Next, we will look at image processing in the Mathematica software.

Exercise.
Using Mathematica software:

1. Save the image in the img variable.
2. Using the `ImageEffect` function decolorize the original image while maintaining the contrast. Save the result in the variable `imgGrayscale`.
3. Convert the newly created image to an array of pixel values. And convert the result of the array of pixel values to an image again.

Solution.

1. `img = [image]`
2. `imgGrayscale = ImageEffect [img, "Decolorization"]`
3. We convert the image to an array of pixel values.

```
ImageData[imgGrayscale][[{1, 2}]];
ImageData[img][[{1, 2}]];
```

We will convert an array of pixel values to an image.

```
ArrayPlot[ImageData [imgGrayscale],
ColorFunction -> GrayLevel]
```

Exercise.

Using Mathematica software:

1. Save the image to the cimg variable.
2. Use the GraphicsRow and RGBColor functions to decompose the image from the cimg variable into three RGB channels. RGBColor[r,g,b] represents a color in the RGB color space with red, blue, and green components.
3. Let's demonstrate the useful ImageHistogram function. We can estimate the pixel density in the color space using ImageHistogram[image]. ImageHistogram[image] will plot a histogram of pixel levels for each channel in the image. Apply the ImageHistogram function to a color image and a grayscale image.

Solution.

1. `cimg = [image]`
2. Decomposition into three RGB channels.

```
GraphicsRow[Prepend[
 MapThread[ArrayPlot[ImageData[#1],
 ColorFunction -> #2] &,
 {ColorSeparate[cimg], {RGBColor [#, 0, 0] &,
 RGBColor [0, #, 0] &, RGBColor [0, 0, #] &}}],
cimg]]
```

3. Graphical representation of an image using a histogram.

```
GraphicsRow[{img, ImageHistogram[img, FrameTicks ->
{Automatic, None}, AspectRatio -> 1]}]
```

```
GraphicsRow[{imgGrayscale, ImageHistogram[imgGrayscale,
FrameTicks -> {Automatic, None}, AspectRatio -> 1]}]
```