

# Report of ECI 2019 Course: “ Introduction to Steganography and Watermarking “

## Assignment E.316-N

Acha Francisco, Cardenas Rodrigo and Castagna Franco.

## 1 Introduction

Steganography is the procedure of insert information inside a data source without changing its perceptual quality. Digital steganography uses digital data sources as a cover for hidden information. Examples of digital covers are digital text files, image files and sound files among others.

In particular for digital image based steganography the pixel intensity is usually used for encoding information but other approaches are also widely used such as embedding information in the frequency domain [1].

There are available many software tools to perform digital image-based steganography. In this report, three steganographic tools that hides text into a digital image were chosen to perform an assessment in terms of imperceptibility of the stego-image, capacity and robustness.

## 2 Materials and methods

### 2.1 Dataset

Since we want to evaluate performance of steganographic tools that hide text into an image, a image dataset is needed. We built a dataset containing images 20 of four types: N-type (landscapes and open nature), S-type (still life), P-type (portraits) and T-type (text). The complete dataset is then 80 images in total. N, S, P-type images were obtained and selected from Google images search engine queries. Namely, keywords for queries were *landscapes*, *still life* and *portrait* respectively. Right usage for the images was selected such that results were labeled for noncommercial reuse, and size of the images was set in medium. Text images were collected from research papers by exporting pages as jpeg images. Table 1 summarizes some basic features of the dataset used such as mean image size and mean file size. All the images in the dataset were stored as jpeg format. Figure 1 shows some examples of images of each type. The text to be embedded was 472 bytes in size and had 466 characters and 89 words <sup>1</sup>.



Figure 1: Two examples for each type from the built dataset are shown. On first column from left, type N images (landscape) are displayed. On second column type P (portrait) images are shown. Third and fourth columns shows type S (still life) and type T (text) images respectively.

---

<sup>1</sup>The text comprises the first three stanzas of the celebrated Argentine text from José Hernández: Martín Fierro.

Table 1: Dataset features.

Feature	Image type			
	N (landscapes)	S (still life)	P (portrait)	T (text)
mean height [pixels] $\pm$ std	600 $\pm$ 100	600 $\pm$ 100	700 $\pm$ 200	1500 $\pm$ 100
mean width [pixels] $\pm$ std	900 $\pm$ 200	800 $\pm$ 200	700 $\pm$ 200	1100 $\pm$ 200
mean size [kBytes] $\pm$ std	200 $\pm$ 100	150 $\pm$ 70	120 $\pm$ 50	300 $\pm$ 100

## 2.2 Description of selected software

### 2.2.1 *OutGuess* (v. 0.2)

*OutGuess* is a universal steganographic tool that allows the insertion of hidden information into the redundant bits of data sources. The nature of the data source is irrelevant to the core of *OutGuess*. The program relies on data specific handlers that will extract redundant bits and write them back after modification. Currently only the PPM, PNM, and JPEG image formats are supported, although *OutGuess* could use any kind of data, as long as a handler were provided. *OutGuess* uses a generic iterator object to select which bits in the data should be modified. A seed can be used to modify the behavior of the iterator. It is embedded in the data along with the rest of the message. By altering the seed, *OutGuess* tries to find a sequence of bits that minimizes the number of changes in the data that have to be made. A bias is introduced that favors the modification of bits that were extracted from a high value, and tries to avoid the modification of bits that were extracted from a low value. Additionally, *OutGuess* allows for the hiding of two distinct messages in the data, thus providing plausible deniability. It keeps track of the bits that have been modified previously and locks them. A (23,12,7) Golay code is used for error correction to tolerate collisions on locked bits. Artificial errors are introduced to avoid modifying bits that have a high bias.

### 2.2.2 *StegHide* (v. 0.5.1)

*StegHide* is an open source steganographic software that allows hide text using image or sound files as covers. It supports JPEG, BMP, WAV and AU file formats as cover files. *StegHide* performs steganography by means of a graph-theoretic approach. Data to be embedded is compressed and encrypted. Then a pseudo-random sequence of positions of pixels is created. On this positions secret data will be embedded. Then a graph-theoretic matching algorithm finds pairs of positions on the remaining pixels such that exchanging their values has the effect of embedding the corresponding part of the secret data. If there are not enough pixels with values that can be used to embed the data by exchanging, values are overwrote. This way, most of the embedding is done by exchanging pixel values and then the first-order statistics is marginally changed. A passphrase must be provided by the user for encryption and pseudo-random generator initialization. The same passphrase must be provided for data extraction from stego-file. The default encryption algorithm is Rijndael with a key of 128 bits although others are available as well.

### 2.2.3 *SilentEye*

*SilentEye* is a cross-platform steganographic software that allows to hide messages into pictures or sounds. It is free and it is open source. It can be used in Microsoft Windows, Mac OS X and Linux. This software supports JPEG, BMP and WAVE format cover files. It provides too, with all its versions, a simple graphical interface and from version 4.4 the option to be used through the command line. In *SilentEye* you can drag and drop to encode and decode data. The encoding window allows you to choose encoding format, output image's quality and other settings. You can have a .txt or any other file ready and merge it directly with the cover file or write a message directly into the program in order to hide it inside the file.

This tool informs us the number of bytes available to hide in the selected cover image. *SilentEye*

also can encrypt the message before hide it into the cover file.

## 2.3 Benchmarking

Some criteria and metrics needs to be established in order to to benchmark the selected software. In this section metrics for imperceptibility assessment are presented as well as criteria regarding capacity of storage for hidden data and tests for robustness evaluation.

### 2.3.1 Imperceptibility

Imperceptibility is a measure of how much the stego-file looks like the original cover file. Is a critical attribute of a steganography system. Having a poor imperceptibility level will make it easier for an eavesdropper to detect the presence of a secret message. For imperceptibility assessment we have selected two widely known metrics namely, Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM) [2].

### 2.3.2 Capacity

Capacity refers to the size of the hidden secret data that a steganographic system is able to embed into to the cover medium. Capacity of a steganographic system depends on the method used to hide data. Hence, different systems can embed different amount of data given the same cover. Capacity for digital image-based steganography can be computed as [1]:

$$C = \frac{P_{secret}N}{P_{cover}}B \quad (1)$$

where  $P_{secret}$  is the number of secret pixels,  $N$  is the number of bits per pixel,  $P_{cover}$  is the number of pixels in the entire cover image and  $B$  is the number of image channels, which can be 3 for RGB representation or 1 if the cover image is represented as grayscale.

### 2.3.3 Robustness

Robustness of a steganographic medium is the property of keep the secret information undestroyed under transformation or tampering of the stego-file. Sistematic assessment of robustness can be done by applying known transformations to the stego-file and evaluating the quality of the hidden data recovered from it. In this work robustness for each steganographic tool was tested under five tampering scenarios: overwriting of 10% of pixels to zero; add of gaussian noise with  $\mu = 0$  and  $\sigma^2 = 2.5 \cdot 10^{-3}$ ; horizontal stretching by 30%; vertical mirroring; and  $20^\circ$  counter clock-wise rotation.

## 3 Results

### 3.1 Imperceptibility

In Figure 2 and Figure 3 results for PSNR and SSIM are respectively shown.

### 3.2 Capacity

Each software provides an option to evaluate the capacity for a given image. For the dataset used, capacity was obtained relative to cover image size and the mean was computed. This way, a capacity per size of cover image es estimated. Results are shown in Table 2.

### 3.3 Robustness

None of the three softwares have shown robustness at all. Any of the five transformations mentioned on Section 2.3.3 made unable to recover the hidden data.

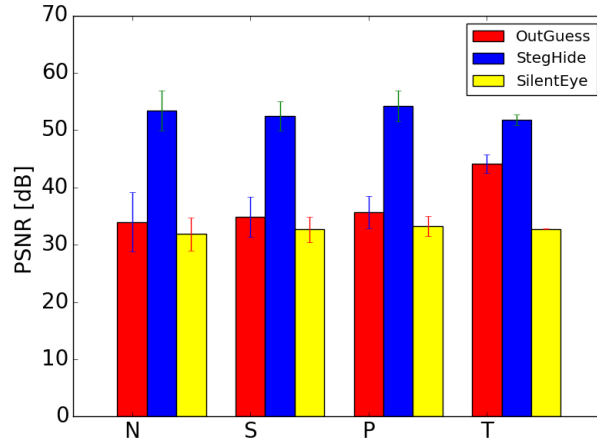


Figure 2: Peak Signal to Noise Ratio for each software and each image type.

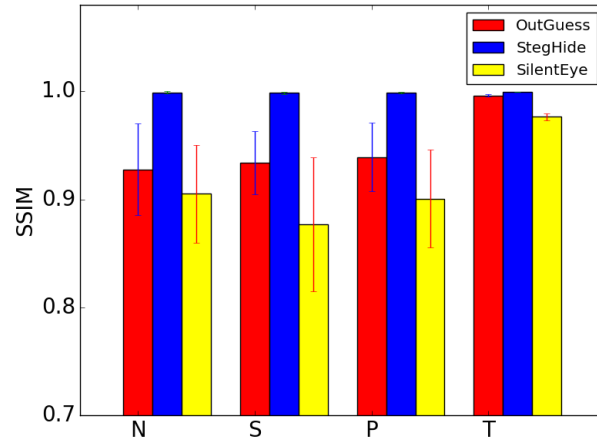


Figure 3: Structural Similarity for each software and each image type.

Table 2: Capacity results for each software.

Software	Capacity per kBytes of cover image [kBytes]
OutGuess	0.10
StegHide	0.05
SilentEye	0.01

## 4 Discussion and Conclusion

Imperceptibility for every tested software was not affected when N, S and P image types were used as cover. However, *StegHide* had the best overall performance except for text images, where each software had a similar performance. It is striking that no software was robust in preserving the integrity of the hidden text, after altering the image in small proportions. An attacker could easily transform the stego-image and it would be impossible to recover the information. We have tested non exhaustively if reducing the length of the text to be hidden could overcome this issue. Only on the case of *OutGuess*, a reduction of the embedded text by 80% enabled extraction of hidden data on altered stego-files. But further tests should be carried out to confirm if this threshold is accurate or if even this occurs as well on the remaining steganographic tools. Regarding capacity, we observed that *OutGuess* is recommended to work with large messages. We see as a disadvantage for these tools to have restrictions for input image formats. This sometimes forces the image to be

converted to another image format, and causes loss of information. Since some of these tools are open source, it would be interesting, as a future work, to improve these tools in order to work with more image formats.

## References

- [1] F. Shih *Digital Watermarking and Steganography: Fundamentals and Techniques (Second Edition)* 2017. CRC Press.
- [2] T. Rabie, M. Baziyad, T. Bonny, R. Fareh *Toward a Unified Performance Metric for Benchmarking Steganography Systems* 2019. Journal of Circuits, Systems and Computers. World Scientific Publishing Company.