UDC: 004.72.056.52:003.27:004.438
DOI: 10.15587/2519-4984.2022.261051

EDUCATIONAL EXAMPLE OF MASKING TEXTUAL INFORMATION IN A PHOTOGRAPHIC SIGNAL

Mykola Holovin, Nina Holovina

The paper presents a steganographic method of masking textual information in photo files. Concealment is implemented in Python. The introduction of individual letters of the text into the image is carried out by the method of "least significant bit". The program can be used for both educational and practical purposes. The Pillow graphics library was used to implement the program. This is not a specialized library for steganographic needs. The use of this library makes it possible to visualize the mechanism of hiding information in the lessons, while the conciseness of the program code gives the possibility to demonstrate it in the classroom setting. It is also important for educational purposes, that working within the Pillow library allows you to see the state of an empty and filled container at the level of individual bits. To assess the practical value of the program, it was tested with texts of different lengths and with containers (photographs) of various kinds. The experiment showed the correct reproduction of texts. Careful visual examination of the empty and correspondingly filled containers (photographs) revealed no differences or suspicions of text bookmarks. Of course, if the party who intercepted the masked message has guesses about how the text is hidden, then this text is easily revealed. Therefore, it is obvious, that the use of the program for practical purposes requires additional manipulations in the code, in particular related to the order of implementation of the text and the choice of location.
It is also desirable to additionally encrypt the text with at least a simple method. Such encryption is possible with the usage of a separate program. Analysis of photographs and manipulation with them at the level of individual bits also has educational value in terms of disclosing a method of capturing the corresponding physical signal. The latter gives an explanation of the methods of encoding static images, noise level, the magnitude of the useful physical signal, and the limits of sensitivity of human vision
Keywords: *Python, Pillow, steganography, hiding, masking information in file, educational example*

How to cite:
Holovin, M., Holovina, N. (2022). Educational example of masking textual information in a photographic signal. ScienceRise: Pedagogical Education, 4 (49), 24–28. doi: http://doi.org/10.15587/2519-4984.2022.261051

1. Introduction

The Internet, as a global information network, makes information easy to transmit through open communication channels. This transfer can take place anywhere in the world. However, unfortunately, there is a wide range of possibilities for intercepting messages. Therefore, there is a need to transmit important information in encrypted or hidden form, and preferably in both versions.

The research of original steganographic methods of hiding information in media files has high theoretical and practical importance. The development of such subjects as programming, steganography, cryptography should be based on current topics. This increases students' motivation to study. When studying programming, in particular, when mastering work with arrays, files, terms, it is interesting to work in the direction of encoding, encryption and hiding information. On the other hand, when studying physics and steganography, it is interesting to have concise, transparent applications of software codes of mechanisms for introducing acoustic and visual physical signals into the relevant media files. It is also useful to have simple applications of software codes for coding, encryption and hiding mechanisms.

2. Literature Review

A brief history of steganography, its scope and its relationship with cryptography are considered in [1]. Here, the most popular steganography algorithms are discussed and some of the tools and software used are demonstrated.

Good systematic publications in the field of steganography are works [2, 3]. The basic steganographic methods of hiding confidential data in audio and graphic computer files are considered here, the problems of stability, bandwidth and reliability of the hidden data exchange channel are systematically considered. Also there it is possible to find the results of information-theoretical research on the problems of concealment of information.

The analysis of estimates of the level of popularity of steganography is interesting. By analyzing a large set of data, such an analysis was performed [4]. The prospects of this approach to information protection are shown.

High-quality research papers [5, 6], written by leaders in the field of steganography of digital media, in a simple way give an idea of the latest trends in digital steganography, principles, algorithms, fundamental theories, methodologies of practical design of modern steganographic tools. A somewhat restored and modified approach to the known methods of cryptography and steganography in accordance with the new dangers can be found in [7].

3. Goal and Objectives of the Research

The goal of this work is to implement a simple software code for hiding textual information in graphic files by means of the Pillow library of the Python language for using this code for practical and educational purposes.

To achieve this goal, the following objectives were set:

1. Create a training program that could hide information in electronic images. Use the Python language, the Pillow graphics library.

2. Optimize and test it with texts of different lengths and with containers (photographs) of different types.

3. Analyze the possibilities of using the program for educational and practical purposes.

4. Materials and methods of research

In this work, hiding "secret" text in an image file (photo) is implemented using the popular algorithm LSB (Least Significant Bit). The "least significant bit" is the bit of the smallest bit in the binary representation of the number.

The essence of this *steganographic method* is to replace the last significant bits of the byte, container (image, audio or video) with the bits of the hidden message. The difference between an empty and a filled container should not be noticeable to the human senses. This idea was tested by the authors when implementing the concealment of textual information in the sound file in Python using the wav library [8].

The idea of the method of hiding the text of the message in the electronic picture, which is implemented in this work, is as follows. The letter codes of the text of the secret message are converted into a bit array. Next, the individual values of this bit array are implemented sequentially in the individual base colors (red, green, blue or RGB) of each pixel of the container in the order they follow.

Each base color (one of three) of an individual pixel is encoded by a byte cell (a number between 0 and 255). Then it is clear, that one pixel of the container can be filled with three bits from the just mentioned bit text array. Note that the base colors are three (RGB) and it is possible to use only the lower bit of each color. Table 1 shows how some of the commonly used colors are encoded.

Table 1

Coding of colors

R	G	B	Color Name	R	G	B	Color Name
0	0	0	Black	80	208	255	Light Blue
255	255	255	White	0	32	255	Blue
224	224	224	Light Gray	96	255	128	Yellow-Green
128	128	128	Gray	0	192	0	Green
64	64	64	Dark Gray	255	224	32	Yellow
255	0	0	Red	255	160	16	Orange
255	96	208	Pink	160	128	96	Brown
160	32	255	Purple	255	208	160	Pale Pink

The Python Pillow graphics library [9, 10] is interesting for steganography because it allows to access individual pixels of the image. In particular, the library allows to change the base colors of individual pixels. It is clear, that this library was created to work with graphics in Python, not for steganography. However, access to individual pixels can be used successfully to hide information.

5. Results of the research

Actions of formal logics of concealment of information

Suppose, for example, the decimal number 130 is the value of the level of redness of the color of a pixel. The binary code of this number looks like 10000010. The smallest significant bit is 0. When the next bit from the bit text array of the bookmark is inserted into the container, this bit either changes to 1 or remains 0. After all, the LSB algorithm replaces the lower bit of each byte of image with one byte of “secret” message. Table 2 shows a scheme of hiding of information.

Manipulation of bits in LSB, shown in the figure, is quite simple, but has two stages.

In the first stage, between each byte of the graphics container and the bit mask 11111110 there is a bitwise logical action "AND", which in Python is denoted by "&". It resets to 0 all the least significant bits of the graphics container bytes.

In the second stage, a logical bit operation "OR" is performed between each modified byte of the container and the bit mask 0000000 [0/1]. Where the least significant bit of the container byte can take the value 0 or 1 from the secret message. Bitwise logical action "OR" is denoted in Python "|". Below, it is possible to see the code for embedding a text message in a graphic file (photo).

Table 2

Scheme of hiding of information

color	RGB values for adjacent pixels								action	254 in binary form									Result of bitwise OR								action	Bitcode of letter N in LSB column									Result of bitwise AND								
R	1	0	0	0	0	0	1	0	&	1	1	1	1	1	1	1	0	=	1	0	0	0	0	0	1	0	|	0	0	0	0	0	0	0	0	=	1	0	0	0	0	0	1	0
G	1	0	0	0	0	0	1	1	&	1	1	1	1	1	1	1	0	=	1	0	0	0	0	0	1	0	|	0	0	0	0	0	0	0	1	=	1	0	0	0	0	0	1	1
B	1	0	0	0	0	1	0	0	&	1	1	1	1	1	1	1	0	=	1	0	0	0	0	1	0	0	|	0	0	0	0	0	0	0	0	=	1	0	0	0	0	1	0	0
R	1	0	0	0	0	1	0	1	&	1	1	1	1	1	1	1	0	=	1	0	0	0	0	1	0	0	|	0	0	0	0	0	0	0	0	=	1	0	0	0	0	1	0	0
G	1	0	0	0	0	1	1	0	&	1	1	1	1	1	1	1	0	=	1	0	0	0	0	1	1	0	|	0	0	0	0	0	0	0	1	=	1	0	0	0	0	1	1	1
B	1	0	0	0	0	1	1	1	&	1	1	1	1	1	1	1	0	=	1	0	0	0	0	1	1	0	|	0	0	0	0	0	0	0	1	=	1	0	0	0	0	1	1	1

Program for hiding text and mechanism of its functioning. The program starts with connecting the PIL library to work with graphic files.

Then the graphic file container and the text itself, which should be further hidden in the graphic file, are loaded.

```
from PIL import Image,ImageDraw          #adding the library PIL and import from it  Image,ImageDraw
image = Image.open("les-ish-.png")       # uploading graphic file of the container
image.show()                             # extraction of the image on the screen
draw = ImageDraw.Draw(image)             # preparing to transformation image image
file = open('secret_text.txt','r',encoding="utf-8")  # opening text file
text=file.read()                         # uploading text into the row variable text
text=text+'#'                            # adding at the end of the text the sign #
print(text);                             # extraction of text from variable text on the screen
file.close()                             # closing the text file
```

After connecting the library and downloading the necessary files in the program, the preparatory steps necessary to mask the text in the image file take place.

There is a conversion of text into a bit array, determining the width and height of the image (container frame), and unloading the value of pixels in the pix array.

```
bits_text=list(map(int,''.join([bin(ord(i)).lstrip('0b').rjust(16,'0') for i in text])))
width = image.size[0]                    # defining the width of an image.
height = image.size[1]                   # defining the height of the image
pix = image.load()                       # defining the indicator of pixels in the massive pix
```

Next, scan the image for height and width, determine the degree of redness (R), green (G), blue (B) color of each pixel. Embedding the next bits of text in the corresponding bytes R, G, B of the next pixels.

```
N=0                                      # providing the variable N (text bits counter) indicator 0
for i in range(width):                   # scanning the image according to width
  for j in range(height):                # scanning the image according to height
    R = pix[i,j][0]                      # defining the level of redness of the color of the pixel
    G = pix[i,j][1]                      # defining the level of greenness of the color of the pixel
    B = pix[i,j][2]                      # defining the level of blueness of the color of the pixel
    if len(bits_text)>N:                 # if the amount of bits of text does not exceed N
      R=(R&254)|bits_text[N]             # implementation of next bit of text in LSB R
      N=N+1                              # number of the next bit
    if len(bits_text)>N:                 # if the number of text bits does not exceed N
      G=(G&254)|bits_text[N]             # implementation of the next bit of text in LSB G
      N=N+1                              # number of the following bit
    if len(bits_text)>N:                 # implementation of the next bit of the text
      B=(B&254)|bits_text[N]             # implementation of the next bit of the text in LSB B
      N=N+1                               # number of  the following bit
    draw.point((i,j),(R,G,B))            # implementation of the current R,G,B in the image
```

At the end of the program, the filled graphic container is displayed and saved together with the text.

```
image.show()                             # display image image on the screen
image.save("les-3-.png", "PNG")          # saving the container along with the text
```

Direct implementation of the text, as a set of bits, in the graphic file occurs bit by bit in each individual current graphic byte. It is clear, that algorithmically it is organized in the form of two nested cycles. Stopping the introduction of bits of text in the image file after their completion is realized by the appropriate branching. Below, there is an arithmetic expression that realizes the implementation of the next bit of text in the red color of the next pixel.

```
R =(R&254)|bits_text[N] # introducing the next bit of text into the red color of the next pixel
```

You can see the use of bitwise operation "&", which performs the role of logical operation "AND" but in the bit information dimension. The operator "|" works similarly in bit space. Its meaning is the logical action of "OR". In the expression, the number 254_{10} plays the role of the bit mask 11111110_2, which was mentioned above, since $254_{10}= 11111110_2$.

The ability to observe the process of hiding the text is opened bit by bit, if directly behind each of the branches if len (bits_text)> N:

display a text bit, a numeric color value (either R or G or B);

color & 254;

(color & 254) | bit.

This feature is quite valuable if you use the program as an application in a lecture when demonstrating its work live through a projector when visualizing the process of hiding information in an image file.

The program for removing the hidden text and the mechanism of its work. To extract a text from an image file, run the code below. Like the previous one, this program starts with connecting the Python Pillow library to work with the image file. Next the direct download of the image file happens, which is then correlated with the variable image. This is where the text is hidden in the image file. To select this text, transform it into an array of pix pixels and start the extracted list to accumulate bits of text that will be extracted from the container image.

```
from PIL import Image, ImageDraw   # adding PIL library and import from it Image,ImageDraw
image = Image.open("les-3-.png")   # uploading graphic file in the image
image.show();                      # showing image on the screen
width = image.size[0]              # defining width of the image.
height = image.size[1]             # defining height of the image
pix = image.load()                 # transformation of file to list of pixels pix
extract=[]                         # opening the list extract
```

Next, all the least significant bits from the graphic bytes are directly extracted into the extracted variable, with the prospect of combining them into letter codes in the text variable. It is clear, that this process is implemented by two cycles of scanning the image in width and height.

```
for i in range(width):                          # scanning width of the image
  for j in range(height):                       # scanning height of the image
    R = pix[i,j][0]                             # defining redness of the color pixel
    G = pix[i,j][1]                             # defining greenness of the color of the pixel
    B = pix[i,j][2]                             # defining blueness of the color of the pixel
# extracting from R,G,B and connecting to an end of variable extract of the current bit
    extract+=[R&1]                              # extraction from R current LSB bit and attaching it to extract
    extract+=[G&1]                              # extraction from G current LSB bit and attaching it to extract
    extract+=[B&1]                              # extraction from B current LSB bit and attaching it to extract
# transformation of bit massive extract to text
text = ""
for i in range(0,len(extract),16)
   text = text+"".join(chr(int("".join(map(str,extract[i:i+16])),2)))
decoded = text.split("#")[0]                    # cutting the text to the delimiting character
print("successfully decoded: "+decoded)         # printing the extracted text
sound.close()
```

Practical significance of hiding information

Tests of programs with texts of different lengths and with graphic containers of different kinds were realized. The experiment showed the correct reproduction of texts, written in both Latin and Cyrillic. The authors did not see any visual differences between empty and filled containers. An experiment was also conducted with the introduction of text into a picture, which was a blank white sheet. Signs of text attachments in the form of any color anomalies are not seen.

However, it should be noted, that if the party who intercepted the masked message has guesses about the fact of bookmarking and the method of bookmarking, the text that was hidden in the manner, described above, is easily removed. Therefore, the use of the program for practical purposes requires additional manipulations in the code, including those related to the order, density of text implementation and the choice of location. It is also desirable to additionally encrypt the text at least with a simple method. Such encryption is possible with a separate program.

Educational significance of the presented program. During creation of the program, a commonly used library was used to work with graphic files. The conciseness of the program code and the use of the Pillow library make it possible to contrast the mechanism of hiding information in the classroom. Thus, in classes on cryptography and steganography, such a demonstration is possible, both in the relevant lecture session and in the process of laboratory work. In programming classes, the mechanism of concealment is well demonstrated live in the process of creating a program, its debugging and testing. It is also important for educational purposes, that working within the Pillow library allows you to see the status of an empty and filled graphics container at the level of individual bytes of pixel colors.

Analysis of the graphic file and its manipulation at the level of individual bits also has educational value in the sense that it gives an idea of the noise level and the magnitude of the useful physical signal as well as the limits of sensitivity of human vision.

The authors argue that the improvement of methodological approaches to practical learning activities in programming is impossible without a real immersion in programming itself. On the other hand, the mechanisms of hiding information are best learned in their direct software implementation.

Research limitations. Research is limited to educational goals, although it illustrates the mechanisms of professional programs.

Prospects for future research. The forerunners of this work were study [8], which considered the concealment of information in the sound signal and study [9], which considered the concealment of information in the electronic picture in another way. In the near future it is planned to develop a concise program for hiding information in video files. This program, like the others just mentioned, will be used for educational purposes in courses of programming, steganography and cryptography.

6. Conclusions

1. Implemented a simple program that allows to hide text information in an image file. Control of the filled and not filled graphic container does not reveal differences. Filled and empty containers are of the same size. The program is interesting for educational purposes due to the accuracy and transparency of the program code. The mechanisms for hiding information in an image file are easily visualized here. The program can be used as an application in a lecture session to demonstrate its work live through a projector. This task is also useful in its implementation in the practical lessons.

2. Analysis of the picture as a physical signal at the level of individual bits, which opens when using the program, also has significant educational value. Manipulations with graphics at such a low level give an idea of the noise level of the sensor when recording the signal, the amplitude and frequency of the useful physical signal, the limits of sensitivity of human vision and the resources to hide in the signal extraneous information.

3. The use of the program for practical purposes requires additional manipulation of the code, in particular, related to the order and density of text implementation, with the choice of its location in the file. It is also desirable to additionally encrypt the text at least with a simple method.

Conflicts of interest

The authors declare that they have no conflicts of interest.

References

1. Bailey, K., Curran, K. (2014). Steganography: The Art of Hiding Information. CreateSpace Independent Publishing Platform.

2. Riabko, B. Ia., Fionov, A. N. (2013). Osnovy sovremennoi kriptografii i steganografii. Moscow: Goriachaia liniia Telekom, 232.

3. Konakhovych, H. F., Prohonov, D. O., Puzyrenko, O. Yu. (2018). Komp'iuterna stehanohrafichna obrobka y analiz multymediinykh danykh. Kyiv: Tsentr navchalnoi literatury, 558.

4. Hegarty, M., Keane, A. (2018). Steganography. The World of Secret Communications, 88.

5. Hassabalah, M. (2020). Digital Media Steganography: Principles, Algorithms, and Advances. Academic Press. doi: http://doi.org/10.1016/c2018-0-04865-3

6. Tanna, S. (2020). Codes, Ciphers, Steganography & Secret Messages. Answers Limited, 263.

7. Wyner, P. (2022). Disappearing Cryptography. Morgan Kaufmann, 295.

8. Holovin, M. B., Holovina, N. A. (2021). Navchalnyi pryklad maskuvannia informatsii v akustychnomu syhnali. Naukovi zapysky Berdianskoho derzhavnoho pedahohichnoho universytetu. Seriia: Pedahohichni nauky, 2, 203–211. Available at: https://evnuir.vnu.edu.ua/handle/123456789/19745

9. Holovin, N., Holovina, N., Yatsiuk, S., Sachuk, Y. (2020). Protection of information steganographically in python by means of the pillow graphic library. Computer-integrated technologies: education, science, production, 40, 110–115. doi: http://doi.org/10.36910/6775-2524-0560-2020-40-17

10. Pillow. Available at: https://pillow.readthedocs.io/en/stable/

Received date 06.06.2022
Accepted date 05.07.2022
Published date 29.07.2022

Nina Holovina*, PhD, Associate Professor, Department of Experimental Physics, Information and Educational Technologies, Lesya Ukrainka Volyn National University, Voli ave., 13, Lutsk, Ukraine, 43025

Mykola Holovin, PhD, Associate Professor, Department of Computer Science and Cybersecurity, Lesya Ukrainka Volyn National University, Voli ave., 13, Lutsk, Ukraine, 43025

****Corresponding author:*** *Nina Holovina, e-mail: ninaholovina@gmail.com*