DOI: https://doi.org/10.36910/6775-2524-0560-2025-59-11
УДК: 004.72.056.52:003.27:004.438
Головін Микола Борисович, к. ф.-м. н., доцент
https://orcid.org/0000-0003-4516-4677
Головіна Ніна Анатоліївна, к. ф.-м. н., доцент
https://orcid.org/0000-0002-1152-1536
Гузачов Дмитро Михайлович, студент
https://orcid.org/0009-0007-3281-602X
Волинський національний університет імені Лесі Українки, м. Луцьк, Україна

ПРОГРАМНА РЕАЛІЗАЦІЯ СТЕГАНОГРАФІЧНОГО ПРИХОВУВАННЯ ТЕКСТОВОЇ ІНФОРМАЦІЇ В ГРАФІЧНОМУ ФАЙЛІ МЕТОДОМ LSB

Головін М. Б., Головіна Н. А., Гузачов Д. М. Програмна реалізація стеганографічного приховування текстової інформації в графічному файлі методом LSB. У роботі представлений код навчальної програми стеганографічного приховування текстової інформації в графічних файлах. Приховування реалізовано методом «наменшого значущого біту» (LSB) мовою Python. Програма може бути використана, як для навчальних, так і практичних цілей. При реалізації програми використана бібліотека Pillow. Ця графічна бібліотека не є спеціалізованою для стеганографічних потреб. Позитивною якістю бібліотеки є можливість візуалізації механізму приховування інформації. Це цікаво для навчальних потреб. Робота в межах бібліотеки Pillow дозволяє на рівні достатньо лаконічного програмного коду побачити механізм приховування інформації на рівні бітів окремих пікселів графіки. Проведено випробовування програми з текстами різної довжини та з графічними контейнерами (фотографіями) різного ґатунку. Експеримент показав коректне впровадження текстів в графічний файл та відповідне вилучення текстів з цих файлів. Дослідження порожніх і відповідних заповнених контейнерів (фотографій) не виявило відмінностей та підозр про текстові закладки. Аналіз фотографій і маніпуляції з ними на рівні окремих бітів має також навчальну цінність в сенсі розкриття способу фіксації відповідного фізичного сигналу. Останнє дає уявлення про способи кодування статичних зображень, рівень шумів, величину корисного фізичного сигналу та межі чутливості людського зору. Представлений код програми може бути використаний як на заняттях з криптографії, так і з практичного програмування.
Ключові слова: стеганографія, захист інформації, приховування інформації, маскування інформації в файлах, статичні зображення, фотографії, мова Python, бібліотека Pillow.

Holovin M., Holovina N., Huzachov D. Software implementation of steganographic hiding of text information in a graphic file using the LSB method. The paper presents the code of a training program for steganographic hiding of text information in graphic files. Hiding is implemented using the "least significant bit" (LSB) method in Python. The program can be used for both educational and practical purposes. The Pillow library was used to implement the program. This graphic library is not specialized for steganographic needs. A positive quality of the library is the ability to visualize the information hiding mechanism. This is interesting for educational purposes. Working within the Pillow library allows us to see the mechanism of information hiding at the bit level of individual graphics pixels at the level of a fairly concise program code. The program was tested with texts of different lengths and with graphic containers (photographs) of different types. The experiment showed the correct introduction of texts into a graphic file and the corresponding extraction of texts from these files. The study of empty and corresponding filled containers (photographs) did not reveal any differences and suspicions of text bookmarks. Analysis of photographs and manipulations with them at the level of individual bits also has educational value in the sense of revealing the method of fixing the corresponding physical signal. The latter gives an idea of the methods of encoding static images, the noise level, the magnitude of the useful physical signal and the sensitivity limits of human vision. The presented program code can be used both in cryptography classes and in practical programming.
Keywords: steganography, information protection, information hiding, information masking in files, static images, photographs, Python language, Pillow library.

Постановка наукової проблеми. Інтернет, як глобальна інформаційна мережа, робить інформацію легкою для передачі відкритими каналами зв'язку. Ця передача може здійснюватися в будь-яку точку Земної кулі. Однак, на жаль, при цьому існує широкий спектр можливостей по перехопленню повідомлень. Тому існує необхідність передачі важливої інформації в зашифрованому або прихованому вигляді, а краще і в зашифрованому і в прихованому.

Актуальними є оригінальні стеганографічні способи приховування інформації в медіа файлах. Освоєння таких освітніх компонент, як програмування, стеганографія, криптографія, має базуватись на актуальних тематиках. Це підвищує мотивацію здобувачів освіти до навчання. При вивченні програмування, зокрема, при освоєнні роботи з масивами, файлами, строками, цікавою є робота в напрямку кодування, шифрування та приховування інформації. З іншого боку, при вивченні фізики та стеганографії цікаво мати лаконічні, прозорі аплікації програмних кодів механізмів впровадження акустичних та візуальних фізичних сигналів у відповідні медійні файли.

Також корисно мати прості аплікації програмних кодів механізмів кодування, шифрування та приховування інформації.

Аналіз останніх досліджень і публікацій. Коротка історія стеганографії, сфери її використання і її зв'язок з криптографією розглянуті в роботі [1]. У цій статті обговорюється призначення стеганографії. Пояснюється як стеганографія пов'язана з криптографією, а також з тим, для чого її можна і не можна використовувати. Крім того, демонструються деякі інструменти та програмне забезпечення, що використовуються в стеганографії. Обговорюються найпопулярніші алгоритми, задіяні у цих інструментах. Пояснюються переваги та недоліки, а також сильні та слабкі сторони використання стеганографії.

Хорошим системним виданням в області стеганографії є робота [2]. Тут розглянуті основні стеганографічні методи приховування конфіденційних даних у звукових та графічних комп'ютерних файлах, системно розглянуті проблеми стійкості, пропускної здатності та надійності каналу прихованого обміну даними. Також тут представлені і результати інформаційно-теоретичних досліджень проблем приховування інформації. Цікавою є аналітика стосовно оцінок рівня популярності стеганографії [3]. Показана перспективність цього підходу до захисту інформації. Дослідницькі роботи [4, 5], що були написані лідерами в галузі стеганографії цифрових медіа, у простому вигляді дають уявлення про останні найсучасніші тенденції цифрової стеганографії, принципи, алгоритми, фундаментальні теорії, методології практичного проектування сучасних стеганографічних засобів. Дещо відновлений і модифікований підхід до відомих способів криптографії і стеганографії, нових небезпек можна знайти в роботі [6].

Метою цієї роботи є реалізація простого програмного коду приховування текстової інформації в графічних файлах засобами бібліотеки Pillow мови Python для використання цього коду в навчальних і прикладних цілях.

Виклад основного матеріалу дослідження. У цій роботі приховування «секретного» тексту в графічному файлі (фотографії) реалізовано за допомогою популярного алгоритму LSB (Least Significant Bit - найменший значущий біт). «Найменший значущий біт» - це біт наймолодшого розряду у двійковому поданні числа. Суть цього стеганографічного методу полягає в заміні останніх значущих бітів байтів, контейнера (зображення чи аудіо) на біти повідомлення, що приховується. Різниця між порожнім та заповненим контейнером не повинна бути відчутна для органів сприйняття людини. Ця ідея була апробована авторами при реалізації приховування текстової інформації в звуковому файлі мовою Python за допомогою бібліотеки wav [7].

Ідея методу приховування тексту повідомлення в електронній картинці, що реалізується в цій роботі, наступна. Коди букв тексту секретного повідомлення перетворюються в бітовий масив. Далі окремі значення цього бітового масиву впроваджуються послідовно в окремі базові кольори (red, green, blue або RGB) кожного пікселя контейнеру в порядку їх слідування.

Кожен базовий колір (один з трьох) окремого пікселя кодується байтовою коміркою (тобто числом в проміжку від 0 до 255). Зрозуміло, що один піксель контейнеру може бути заповнений трьома бітами з щойно згаданого бітового текстового масиву. Враховуємо, що базових кольорів три (RGB) і ми можемо використати тільки молодший біт кожного кольору. В таблиці 1 представлено, як кодуються деякі з широко вживаних кольорів.

Таблиця 1. Коди кольорів

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

Графічна бібліотека Python Pillow [8] є цікавою для стеганографії. Адже, вона дає можливість отримати доступ до окремих пікселів зображення. Зокрема, бібліотека дає можливість змінювати базові кольори окремих пікселів. Зрозуміло, що ця бібліотека створювалась для роботи з графікою в мові Python, а не для стеганографії. Однак, доступ до окремих пікселів можна з успіхом використовувати для приховування інформації.

Дії формальної логіки по приховуванню інформації. Нехай, наприклад, десяткове число 130 є значенням рівня червоності кольору деякого пікселя. Двійковий код цього числа має вигляд 10000010. Найменший значущий біт дорівнює 0. При впровадженні в контейнер чергового біта з бітового текстового масиву закладки цей біт або змінюється на 1 або залишається 0. Адже алгоритм LSB замінює молодший біт кожного байта зображення одним бітом із «секретного» повідомлення.

Таблиця 2. Схема приховування даних

колір	Значення RGB суміжних пікселів								дія	254 в двійковому вигляді									Результат побітового OR								дія	Бітовий код букви N в LSB стовбці									Результат побітового 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
R	1	0	0	0	1	0	0	0	&	1	1	1	1	1	1	1	0	=	1	0	0	0	1	0	0	0	|	0	0	0	0	0	0	0	1	=	1	0	0	0	1	0	0	1
G	1	0	0	0	1	0	0	1	&	1	1	1	1	1	1	1	0	=	1	0	0	0	1	0	0	0	|	0	0	0	0	0	0	0	0	=	1	0	0	0	1	0	0	0

Маніпуляції бітами в LSB, що показані в таблиці 2, досить прості. Однак, вони мають два етапи.

На першому етапі між кожним байтом графічного контейнера і бітовою маскою 11111110 відбувається побітова логічна дія «AND», яка в мові Python позначається «&». Вона скидає на 0 всі найменш значущі біти байтів графічного контейнеру. ***На другому етапі*** між кожним модифікованим байтом контейнеру і бітовою маскою 0000000[0/1] виконується логічна побітова операція «OR». Де молодший біт байта контейнера може прийняти значення 0 або 1 із секретного повідомлення. Побітова логічна дія «OR» позначається в мові Python «|». Нижче представлений код програмного вбудовування текстового повідомлення в графічний файл (фотографію).

Програма приховування тексту та механізм її роботи. Програма розпочинається з під'єднання бібліотеки PIL для роботи із графічними файлами. Далі завантажується графічний файл контейнер та сам текст який надалі має бути прихований у графічному файлі.

```
from PIL import Image,ImageDraw     #підключення бібліотеки PIL
image = Image.open("les-ish-.png") # завантаження графічного файлу контейнера
image.show()                        # виводимо картинку image на екран
draw = ImageDraw.Draw(image)        # підготовка до перетворень image
file = open('secret_text.txt','r',encoding="utf-8") # відкриття текстового файлу
text=file.read()                    # завантаження тексту в рядкову змінну text
text=text+'#'                       # додавання в кінець тексту знаку #
print(text);                        # вивід тексту з змінної text на екран
file.close()                        # закриття текстового файлу
```

Після під'єднання бібліотеки і завантаження потрібних файлів в програмі відбуваються підготовчі дії необхідні для маскування тексту в графічному файлі. Відбувається перетворення тексту в бітовий масив, визначення ширини та висоти зображення (картинки контейнеру), вивантаження значення пікселів в масив pix.

```
bits_text=list(map(int,''.join([bin(ord(i)).lstrip('0b').rjust(16,'0') for i in text])))
width = image.size[0]      # визначення ширини зображення.
height = image.size[1]     # визначення висоти зображення
pix = image.load()         # завантаження значення пікселів в масив pix
```

Далі сканування картинки за висотою та шириною, визначення міри червоності (R), зеленості (G), блакитності (B) кольору кожного пікселя. Впровадження чергових бітів тексту у відповідні байти R,G,B чергових пікселів.

```
N=0                              # надання змінній N (рахівник бітів тексту) значення 0
for i in range(width):           # сканування картинки по ширині
   for j in range(height):       # сканування картинки по висоті
      R = pix[i,j][0]            # визначення міри червоності коліру пікселя
      G = pix[i,j][1]            # визначення міри зеленості коліру пікселя
      B = pix[i,j][2]            # визначення міри блакитності коліру пікселя
      if len(bits_text)>N:       # якщо кількість бітів тексту не перевищує N
         R=(R&254)|bits_text[N]  # впровадженя чергового біту тексту в LSB R
         N=N+1                   # номер наступного біту
      if len(bits_text)>N:       # якщо кількість бітів тексту не перевищує N
         G=(G&254)|bits_text[N]  # впровадженя чергового біту тексту в LSB G
         N=N+1                   # номер наступного біту
      if len(bits_text)>N:       # впровадженя чергового біту тексту
         B=(B&254)|bits_text[N]  # впровадженя чергового біту тексту в LSB B
         N=N+1                   # номер наступного біту
      draw.point((i,j),(R,G,B)) # впровадження поточних R,G,B в draw (в картинку)
```

В кінці роботи програми відбувається вивід на екран заповненого графічного контейнеру та збереження його разом з текстом.

```
image.show()                                # вивід картинки image на екран
image.save("les-3-.png", "PNG")             # збереження контейнеру разом з текстом
```

Безпосереднє впровадження тексту, як сукупності бітів, у графічний файл відбувається побітово почергово в кожний окремий поточний графічний байт. Зрозуміло, що алгоритмічно це організовано у вигляді двох вкладених один в одний циклів. Зупинка впровадження бітів тексту в графічний файл після їх закінчення реалізується відповідним розгалуженням. Нижче представлений арифметичний вираз, що реалізує впровадження чергового біту тексту в червоний колір чергового пікселя.

```
R =(R&254)|bits_text[N]
```

Як вже згадувалось, використання операції «&» виконує роль логічної операції «AND» але в бітовій інформаційній розмірності. Аналогічно спрацьовує в бітовому просторі оператор «|». Його сенс логічна дія «OR». У виразі число 254_{10} грає роль бітової маски 11111110_2, про яку йшлося вище. Адже $254_{10}= 11111110_2$.

Представлена програма надає важливу для навчання криптографії, можливість спостереження процесу приховування тексту в медійний файл побітово. Ця можливість відкривається, якщо в програму добавити програмні фрагменти подібні наданому нижче.

```
if len(bits_text)>N:
    print('(','.join(format(G,'b')),'&','.join(format(254,'b')),')',end='|')
    G=(G&254)| bits_text[N]; N=N+1
    print(''.join(format(bits_text[N], 'b')),"=",''.join(format(G, 'b')))
```

Ця можливість є досить цінною, якщо використовувати програму, як аплікаційну, на лекційному занятті при демонстрації процесу приховування інформації в графічному файлі.

Програма вилучення прихованого тексту та механізм її роботи. Для вилучення тексту з графічного файлу необхідно запустити представлений нижче код. Як і попередня, ця програма починається з підключення бібліотеки Python Pillow для роботи з графічними файлом. Далі відбувається безпосереднє завантаження графічного файлу, який надалі співвідноситься з змінною image. Саме тут знаходиться текст прихований в графічному файлі. Для виділення цього тексту трансформуємо його в масив пікселів pix та започаткуємо список extracted для накопичення бітів тексту, що будуть видобуватись з картинки контейнеру.

```
from PIL import Image, ImageDraw # підключення бібліотеки PIL
image = Image.open("les-3-.png") # завантаження графічного файлу в image
image.show();                    # виводиться картинка image на екран
width = image.size[0]            # визначення ширини зображення.
height = image.size[1]           # визначення висоти зображення
pix = image.load()               # трансформація файлу в список пікселів pix
extracted=[]                     # відкриття списку extracted
```

Далі проводиться безпосереднє вилучення всіх найменших значущих бітів з графічних байтів в змінну extracted з подальшою перспективою на об'єднання їх в коди букв в змінній text.

Зрозуміло, що цей процес реалізується двома циклами сканування картинки по ширині і висоті.

```
for i in range(width):    # сканування картинки по ширині
  for j in range(height): # сканування картинки по висоті
     R = pix[i,j][0]      # визначення міри червоності коліру пікселя
     G = pix[i,j][1]      # визначення міри зеленості коліру пікселя
     B = pix[i,j][2]      # визначення міри блакитності коліру пікселя
     # вилучення з R, G, B та підклейка в кінець змінної extracted поточного біту
     extracted+=[R&1]     # вилучення з R поточного LSB біту та долучення до extracted
     extracted+=[G&1]     # вилучення з G поточного LSB біту та долучення до extracted
     extracted+=[B&1]     # вилучення з B поточного LSB біту та долучення до extracted
# перетворення бітого масиву extracted в текст text
text = "".join(chr(int("".join(map(str,extracted[i:i+16])),2)) for i in
range(0,len(extracted),16))
decoded = text.split("#")[0] # вирізка тексту, що знаходиться до символу роздільника
print("Успішно декодовано: "+decoded) # роздруковка вилученого тексту
```

Практична цінність приховування інформації. Було проведено випробовування програм з текстами різної довжини та з графічними контейнерами різного ґатунку. Експеримент показав коректне відтворення текстів, що були написані, як латиницею, так і кирилицею. Автори не побачили візуальних відмінностей між порожніми і заповненими контейнерами. Проводився також експеримент з впровадженням тексту в картинку, що була чистим білим листом. Ознак текстових вкладень у вигляді яких-небудь кольорових аномалій не побачили.

Однак, необхідно відмітити, якщо сторона, що перехватила замасковане повідомлення, має здогадки про факт закладки та спосіб закладки, то текст, що був прихований описаним вище способом легко вилучається. Тому використання програми у практичних цілях вимагає додаткових маніпуляцій в коді, зокрема пов'язаних з порядком, щільністю впровадження тексту та з вибором місця впровадження. Бажаним є також додаткове шифруванням тексту хоча б простим методом. Таке шифрування можливо і окремою програмою.

Цінність представленої програми для навчальних цілей. При реалізації програми була застосована загальновживана бібліотека для роботи з графічними файлами. Лаконічність коду програми та використання бібліотеки Pillow дає можливість контрастно візуалізувати механізм приховування інформації на навчальних заняттях. Так на заняттях з криптографії і стеганографії така демонстрація можлива, як на відповідному лекційному занятті, так і в процес проведення лабораторної роботи. На заняттях з програмування механізм приховування добре демонструвати вживу в процесі створення програми, її відлагодження та випробовування. Важливо для навчальних цілей і те, що робота в межах бібліотеки Pillow дозволяє на рівні окремих байтів кольорів пікселя побачити стан порожнього та заповненого графічного контейнеру.

Аналіз графічного файлу і маніпуляції з ним на рівні окремих бітів має також навчальну цінність в тому сенсі, що дає уявлення про рівень шумів та величину корисного фізичного сигналу та межі чутливості людського зору.

Висновки та перспективи подальшого дослідження.

- Реалізовано просту програму, що дозволяє приховувати текстову інформацію в графічному файлі. Контроль заповненого і не заповненого графічного контейнера не виявляє відмінностей. Заповнений і порожній контейнер мають однаковий розмір.
- Програма цікава для навчальних цілей завдяки лаконічності і прозорості програмного коду. Механізми приховування інформації в графічному файлі тут легко візуалізується. Програма може бути використана, як аплікаційна на лекційному занятті при демонстрації її роботи вживу через проектор. Корисна ця задача і при її реалізації на лабораторному занятті.
- Аналіз картинки, як фізичного сигналу, на рівні окремих бітів, що відкривається при використанні програми, має також значну навчальну цінність. Маніпуляції з графікою на такому низькому рівні дають уявлення про амплітуду корисного фізичного сигналу, рівень шумів, межі чутливості людського зору. Цікаві також і ресурси для приховування в сигналі сторонньої інформації.
- Використання програми в практичних цілях вимагає додаткових маніпуляцій в коді, зокрема, пов'язаних з порядком та з щільністю впровадження тексту, з вибором його місця розташування в файлі. Бажаним є також додаткове шифруванням тексту хоча б простим методом.

Список бібліографічного опису

1. Kefa Rabah. Steganography-The Art of Hiding Data. Information Technology Journal, 2004. 3 (3): P. 245-269. https://scialert.net/fulltext/fulltextpdf.php?pdf=ansinet/itj/2004/245-269.pdf
2. Г. Ф. Конахович, Д. О Прогонов, О. Ю. Пузиренко, *Комп'ютерна стеганографічна обробка й аналіз мультимедійних даних* [підручник]. К.: Центр навчальної літератури, 2018. 558 с. https://pdf.lib.vntu.edu.ua/books/2019/Konahovich_2018_558.pdf
3. Hegarty, M., and Keane, A. Steganography, The World of Secret Communications. 2018. 88 p.
4. Hassabalah, M. Digital Media Steganography: Principles, Algorithms, and Advances. Academic Press. 2020. https://www.researchgate.net/publication/342612073_Digital_Media_Steganography_Principles_Algorithms_Advances
5. Tanna, S. Codes, Ciphers, Steganography & Secret Messages. Answers Limited. 2020. https://ia601800.us.archive.org/4/items/7-the-code-breaker/7-%20The%20Code%20Breaker.pdf
6. Wyner, P. Disappearing cryptography: information hiding: steganography & watermarking, 2002. 413p. https://search.worldcat.org/title/1024951811
7. Головін М.Б., Головіна Н.А. Навчальний приклад маскування інформації в акустичному сигналі, *Наукові записки Бердянського державного педагогічного університету. Серія: Педагогічні науки.* Вип. 2, С. 203-211, 2021. Doi. https://evnuir.vnu.edu.ua/handle/123456789/19745
8. https://pillow.readthedocs.io/en/stable/

References

1. Kefa Rabah. Steganography-The Art of Hiding Data. Information Technology Journal, 2004. 3 (3): P. 245-269. https://scialert.net/fulltext/fulltextpdf.php?pdf=ansinet/itj/2004/245-269.pdf
2. G.F. Konahovych, D.O. Progonov, O. Yu. Puzyrenko, Computer steganographic processing and analysis of multimedia data [textbook]. K.: Center of Educational Literature, 2018.
3. Hegarty, M., and Keane, A. Steganography, The World of Secret Communications. 2018. 88 p.
4. Hassabalah, M. Digital Media Steganography: Principles, Algorithms, and Advances. Academic Press. 2020. https://www.researchgate.net/publication/342612073_Digital_Media_Steganography_Principles_Algorithms_Advances
5. Tanna, S. Codes, Ciphers, Steganography & Secret Messages. Answers Limited. 2020. https://ia601800.us.archive.org/4/items/7-the-code-breaker/7-%20The%20Code%20Breaker.pdf
6. Wyner, P. Disappearing cryptography: information hiding: steganography & watermarking, 2002. 413p. https://search.worldcat.org/title/1024951811
7. Holovin M., Holovina N. A case study of information masking in the acoustic signal, *Scientific notes of Berdyansk State Pedagogical University*. Series: Pedagogical sciences. Issue. 2, P.203 -211, 2021. doi. https://evnuir.vnu.edu.ua/handle/123456789/19745
8. https://pillow.readthedocs.io/en/stable/