Relazione Progetto Facoltativo

Digital Forensics

Appello del 02 luglio 2024

Tool Steganografico

Bellamacina

Giuseppe Cosimo Alfio

1000030349

INDICE

1. Steganografia
- 1.1. Cos’è la Steganografia
- 1.2. Come funziona la Steganografia
- 1.3. Steganografia nelle immagini
- 1.4. Tecniche steganografiche nelle immagini
- 1.5. Vantaggi e Limiti
- 1.6. Applicazioni

2. Come utilizzare il Tool
- 2.1. Requisiti
- 2.2. Avvio del programma
- 2.3. Utilizzo del programma
  - 2.3.1. Stringhe
  - 2.3.2. File di testo
  - 2.3.3. Immagini
  - 2.3.4. File binari

3. Descrizione tecnica del Tool
- 3.1. Funzioni di avvio e interfaccia
- 3.2. Occultamento di stringhe
- 3.3. Occultamento di file testuali
- 3.4. Occultamento di immagini
- 3.5. Occultamento di file binari e cartelle
- 3.6. Dati di backup

4. Conclusioni

1. Steganografia

1.1. Cos'è la Steganografia

La steganografia è l'arte e la scienza di nascondere messaggi o informazioni all'interno di altri dati apparentemente innocui, in modo tale che la loro presenza non sia rilevabile. Questo termine deriva dalle parole greche "steganos" (nascosto) e "graphia" (scrittura), indicando quindi la pratica di nascondere la scrittura. A differenza della crittografia, che rende un messaggio incomprensibile a chi non possiede la chiave per decifrarlo, la steganografia mira a mantenere l'esistenza del messaggio nascosto segreta.

1.2. Come funziona la Steganografia

Il funzionamento della steganografia si basa sull'inserimento di informazioni all'interno di un medium di copertura (cover medium), che può essere un'immagine, un audio, un video, o anche un testo. Il medium di copertura deve apparire normale e non deve mostrare evidenti segni di alterazione che possano suggerire la presenza di dati nascosti. L'efficacia della steganografia dipende dalla capacità di nascondere il messaggio in modo tale che non sia rilevabile senza conoscenza pregressa o strumenti specifici.

1.3. Steganografia nelle immagini

Uno dei metodi più comuni di steganografia è quello di nascondere dati all'interno delle immagini digitali. Le immagini digitali sono composte da pixel, ognuno dei quali ha valori numerici che rappresentano i colori (e in certi casi la trasparenza (es. png)). Utilizzando tecniche steganografiche, è possibile modificare leggermente questi valori per incorporare informazioni senza alterare visibilmente l'immagine.

1.4. Tecniche steganografiche nelle Immagini

Esistono diverse tecniche per nascondere dati all'interno delle immagini. Di seguito sono riportate alcune delle più comuni:

1. **Least Significant Bit (LSB) Insertion**:
    - La tecnica più comune e semplice. Consiste nel modificare i bit meno significativi dei pixel di un'immagine.
    - Ad esempio, in un'immagine a 24 bit, ogni pixel è rappresentato da 3 byte (uno per il rosso, uno per il verde e uno per il blu). Modificando l'ultimo bit di ogni byte, è possibile nascondere informazioni senza alterare visibilmente il colore del pixel.
2. **Masking and Filtering**:
    - Tecniche più complesse che coinvolgono la manipolazione di porzioni specifiche dell'immagine (come l'applicazione di filtri) per nascondere informazioni.
3. **Transform Domain Techniques**:

    - Utilizzano trasformazioni matematiche (come la Discrete Cosine Transform, DCT) per nascondere dati nelle frequenze dell'immagine, anziché nei valori dei pixel diretti.

4. **Redundant Pattern Encoding**:
    - Utilizza modelli ridondanti e la ripetizione per nascondere i dati, rendendo più difficile la loro rilevazione.

1.5. Vantaggi e Limiti della Steganografia

Vantaggi:

- **Segretezza**: La presenza di un messaggio nascosto non è evidente.
- **Versatilità**: Può essere applicata a vari tipi di media (immagini, audio, video, testo).
- **Combinabilità**: Può essere combinata con altre tecniche di sicurezza, come la crittografia, per aumentare la sicurezza.

Limiti:

- **Capacità**: La quantità di dati che può essere nascosta è limitata dalle dimensioni e dalla qualità del medium di copertura.
- **Rilevabilità**: Con tecniche avanzate di analisi delle immagini, è possibile rilevare la presenza di dati nascosti.
- **Alterazione**: Qualsiasi compressione o modifica del medium di copertura può distruggere i dati nascosti.

1.6. Applicazioni

La steganografia ha numerose applicazioni in diversi campi, tra cui:

- **Sicurezza delle comunicazioni**: Nascondere messaggi in immagini per evitare l'intercettazione.
- **Digital Watermarking**: Protezione del diritto d’autore di immagini e video.
- **Evasione di censura**: Trasmettere informazioni in regimi dove la censura è pesante.
- **Data Integrity**: Verificare l'integrità dei dati.

2. Come utilizzare il Tool

2.1. Requisiti

Il tool è molto semplice da utilizzare, ma prima di avviare il programma è necessario verificare di disporre di certi strumenti:

- **Python 3.9+**: serve a interpretare i comandi del linguaggio Python;
- **Pip**: utile per installare le librerie sotto riportate;
- **Librerie Python**:
  - **Numpy;**
  - **Scipy;**
  - **Pandas;**
  - **Plotly;**
  - **Pillow;**
  - **Matplotlib;**
  - **Jupyter;**
  - **Termcolor;**
  - **Pyfiglet;**
  - **Colorama;**

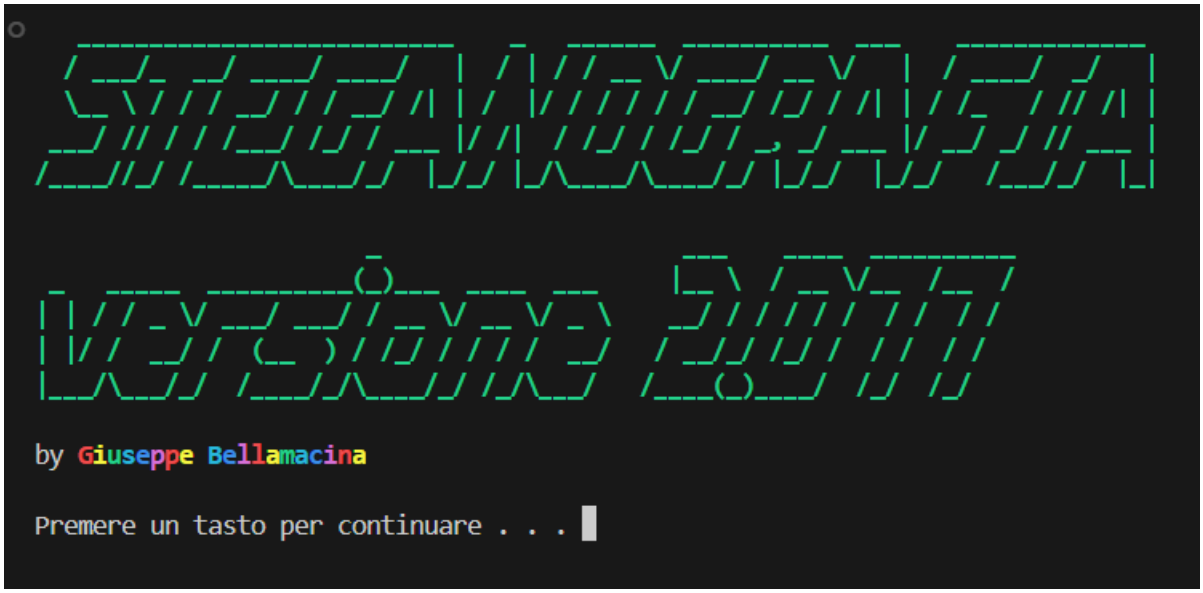
Tutti questi requisiti possono essere ottenuti facilmente su Windows eseguendo il file **lib_install.bat** in allegato.

2.2. Avvio del programma

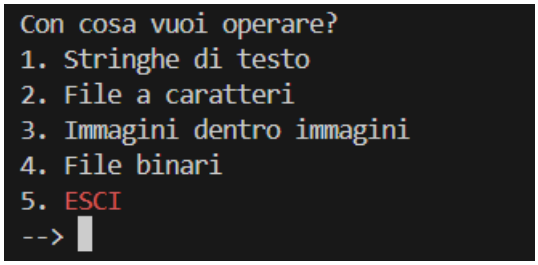
Per avviare il programma si può utilizzare il comando

python steganografia.py

da linea di comando. Oppure si può avviare il programma con IDLE specializzato.



Questa è la schermata d'avvio del programma. Il programma ha un'interfaccia grafica di tipo testuale ma dotata di colori. In questo caso il titolo si è presentato in colorazione verde, ma il colore è casuale ad ogni avvio.



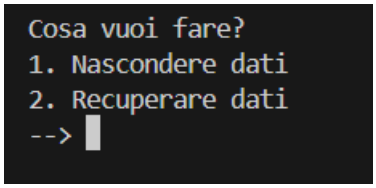
Dopo aver premuto invio, si accede al menu principale. Da qui si possono scegliere le varie modalità di funzionamento. Il programma gestisce anche gli eventuali errori dell'utente.

2.3. Utilizzo del programma

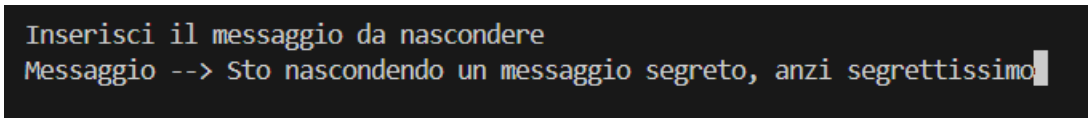
Vediamo di seguito come usare le varie modalità:

2.3.1. Stringhe

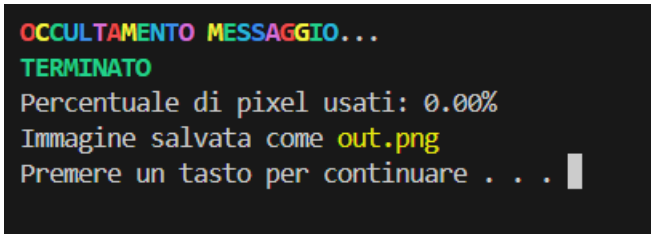
Si preme **1** e si invia.



Si può scegliere se si vuole nascondere o recuperare dei dati. Se si decide di nascondere dei dati viene successivamente richiesto di specificare su quale immagine si vuole nascondere questi ultimi (in questo caso una stringa di testo). Dopo viene chiesto di inserire il messaggio da nascondere.



A questo punto si inserisce il nome dell'immagine di output. Il processo si avvia subito dopo ed è molto veloce.

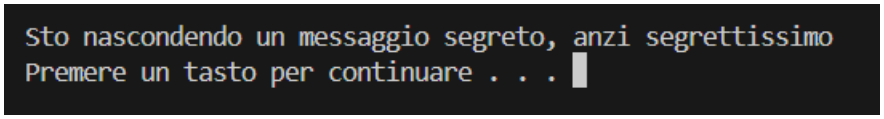


Si noti che il programma informa l'utente sulla percentuale di pixels modificati. Dopo di ciò il programma salva automaticamente i dati necessari a recuperare il messaggio in memoria, ma questi ultimi vengono persi se si chiude il programma. Se si vuole recuperare il messaggio in

un secondo momento è possibile chiedere al programma di salvare questi dati utili al recupero in un file apposito. Si deve specificare il nome del file con questi dati.

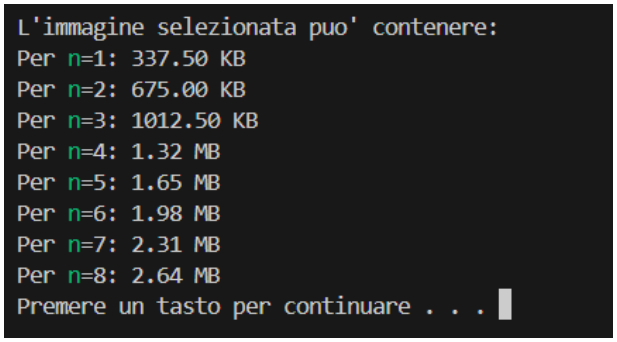
Se invece l'utente dovesse recuperare dei dati il programma chiederebbe se volesse usare dei parametri di backup. Se l'utente accetta il programma chiederà se vorrà usare i dati salvati su un eventuale file di recupero creato alla fine della fase di occultamento dei dati, oppure di utilizzare i dati salvati automaticamente. Questi dati riguardano l'ultima procedura di occultamento eseguita. Se la procedura non è compatibile (es. si sta provando a recuperare una stringa dopo che l'utente ha appena nascosto un'immagine) oppure l'utente non ha ancora occultato nessun tipo di dato (il programma è stato appena aperto), allora il programma lancerà un'eccezione e tornerà al menù.

Se l'utente dovesse scegliere di non utilizzare i parametri di backup, allora il programma chiederà di inserire manualmente i dati necessari al recupero. Questi dati si limitano al solo nome dell'immagine contenente i dati nascosti nel caso di un'immagine su cui è stata nascosta una stringa.



2.3.2. File di testo

Anche in questo caso il programma chiede le intenzioni dell'utente, ma se egli vuole nascondere dei dati, il programma mostra quanti dati è possibile nascondere sulla foto selezionata.



In questo esempio è stata scelta un'immagine di 720 x 1280 pixels da 3 canali di colore RGB. Si noti che il programma dice quanti dati si possono inserire a seconda di quanti bit meno significativi si scelga di utilizzare.

Dopo si deve scegliere il file testuale da nascondere (se è troppo grande il programma lo segnala e ne fa scegliere un altro) ed il nome dell'immagine di output.

A questo punto si devono scegliere i parametri utili al programma per capire come occultare i dati. I parametri in questa modalità sono **div** e **n**, vale a dire la distanza tra un canale modificato ed un altro nell'array formato dalla concatenazione di tutti i canali di tutti i pixels in ordine e la quantità di bit meno significativi da modificare.

L'utente può scegliere di far decidere dinamicamente al programma, il quale darà sempre la priorità alla qualità della foto, oppure potrà scegliere manualmente ogni parametro. Il

programma glieli farà inserire uno alla volta, e per ogni parametro darà sempre la possibilità di fare calcolare il singolo parametro al programma. Se alla fine della procedura manuale di selezione dei parametri, il programma noterà che i parametri scelti non permettono di occultare l'immagine, allora lancerà un'eccezione. Ad esempio, l'utente potrebbe voler caricare un file di 2 MB su un'immagine che può contenere al massimo 2.5 MB se si dovesse scegliere di utilizzare n=8 (non consigliato perché la procedura produrrà un'immagine drasticamente degradata), ma l'utente decide di porre n=1.

Di seguito sono riportate le immagini di output di alcune procedure eseguite con parametri differenti sulla stessa immagine e occultando lo stesso file testuale (l'intero testo della Divina Commedia):

Immagine originale

Modalità automatica
n=2
div=1.23

n=8
div=4.95 (auto)

n=8
div=3

n=2 (auto)
div=1

n=8
div=1

Si noti il caso con n=8 e div=3. La foto assume una tonalità nettamente più rossa poiché il programma itera sui canali di colore dei pixel e ogni tre posizioni modifica tutti i bit, ma così facendo si ritroverà a modificare sempre il canale del rosso. L'ultimo caso invece modifica ogni canale poiché div=1 e in modo completo dato che n=8. L'immagine è infatti fortemente degradata. Anche la terza immagine utilizza n=8, ma poiché l'utente ha scelto di utilizzare un div calcolato dal programma (in questo caso div=4.95), il programma pur rendendo evidente la manipolazione dell'immagine, non la distrugge completamente.

Il programma avvia un countdown e al termine dello stesso, mostra i parametri utilizzati.

```
OCCULTAMENTO FILE...
Elaborate 14753 righe su 14753 (100.0%)
Tempo rimanente: 0 secondi
TERMINATO
Percentuale di pixel usati con n=2 e div=1.2387980321169976: 80.72%
Immagine salvata come out1.png
Premere un tasto per continuare . . .
```

Anche in questo caso si possono salvare i parametri per il recupero e la procedura di recupero dei dati è uguale a quella per le stringhe.

2.3.3. Immagini

SI possono anche nascondere delle immagini dentro le immagini, ma in questo caso si possono fronteggiare dei casi di perdita di dati.

Il programma come sempre chiede all'utente cosa vuole fare e dopo chiede di scegliere una foto su cui caricare una seconda foto. Qui mostra quanti pixels può modificare, non quanti dati può occultare. È stata scelta la stessa immagine grande 720 x 1280 a 3 canali RGB.

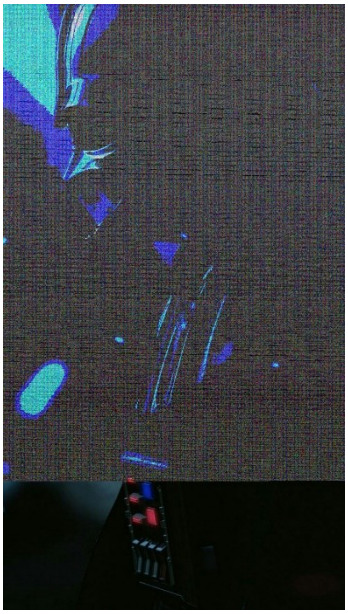
```
L'immagine selezionata puo' contenere:
Per n=1: 345600 pixels da 3 canali
Per n=2: 691200 pixels da 3 canali
Per n=3: 1036800 pixels da 3 canali
Per n=4: 1382400 pixels da 3 canali
Per n=5: 1728000 pixels da 3 canali
Per n=6: 2073600 pixels da 3 canali
Per n=7: 2419200 pixels da 3 canali
Per n=8: 2764800 pixels da 3 canali
Premere un tasto per continuare . . .
```

Il programma chiede di scegliere i parametri per nascondere i dati, ed in questo caso sono:

- **msb**: vale a dire la quantità di bit più significativi da preservare dell'immagine da nascondere;
- **lsb**: vale a dire la quantità di bit meno significativi da modificare dell'immagine su cui occultare la seconda foto;
- **div**: vale a dire la distanza tra un canale modificato ed un altro nell'array formato dalla concatenazione di tutti i canali di tutti i pixels in ordine.

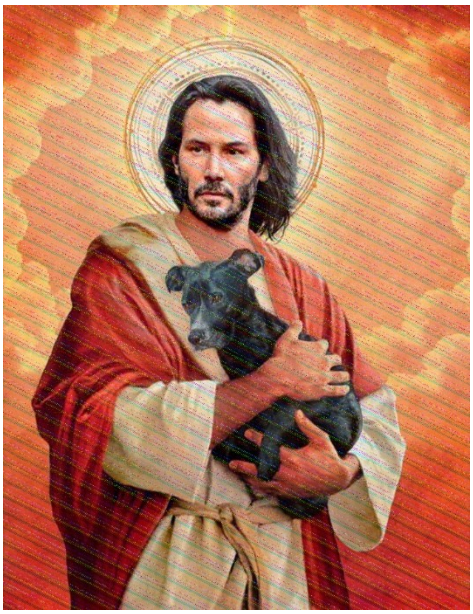
Di seguito sono mostrate alcuni risultati di alcune procedure di occultamento ed estrazioni di immagini da immagini con parametri differenti ma sulle stesse immagini. Le immagini sono quella sopracitata (su cui nascondere i dati) ed un'altra grande 606 x 1280 da 3 canali RGB:


Immagini originali

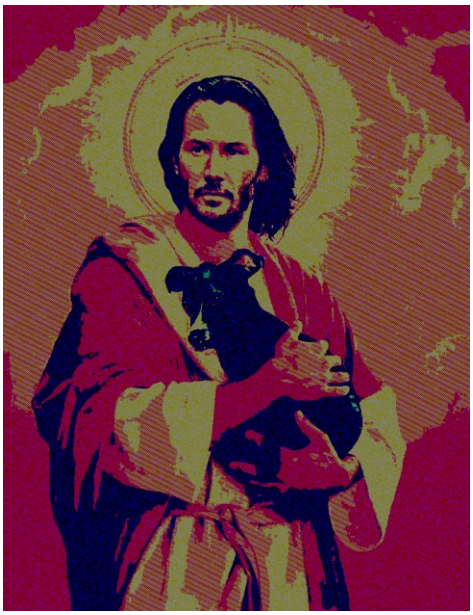



lsb=7
msb=8
div=1

Le due immagini hanno dimensioni molto simili, anche se la prima è più grande della seconda. In questo primo caso si è scelto di sacrificare 7 bit meno significativi su 8 della prima in modo da salvaguardare tutti e 8 i bit della seconda. Si è scelto pure di distribuire le modifiche su ogni elemento dell'array dei canali dei pixels dato che div è 1. Si vede infatti che a seguito dell'estrazione non si è perso neanche un dettaglio dell'immagine nascosta, ma l'immagine usata come nascondiglio per la seconda è stata pesantemente modificata.

Modalità automatica
lsb=6
msb=8
div=1.08

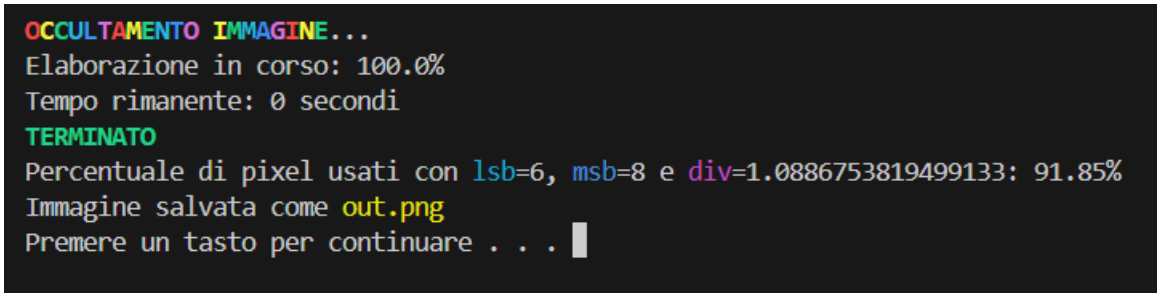
lsb=1
msb=1
div=1.45 (auto)

In questi due casi invece si è scelto inizialmente di utilizzare i parametri automatici, i quali hanno dato priorità alla salvaguardia della seconda foto, ma ciò non è bastato poiché l'immagine estratta è leggermente degradata. Questo poiché nonostante il parametro msb sia 8, il div è di poco superiore a 1 (come già detto le 2 immagini sono molto simili di dimensioni), dunque è altamente probabile che qualche pixel sia stato corrotto nella fase di occultamento. L'immagine di base però è meno degradata rispetto a quella del primo esempio.

Nell'ultimo esempio invece di è scelto di salvare solo il bit più significativo di ogni canale della seconda foto e di modificare il bit meno significativo dei corrispettivi della prima foto. In

questo caso il div è stato scelto in maniera automatica. Qui la foto di base è stata manipolata in maniera impercettibile, mentre la foto estratta è profondamente degradata.

Il programma informa l'utente allo stesso modo della modalità precedente:



E successivamente chiede se l'utente vuole salvare le informazioni necessarie a recuperare i dati appena occultati. Nota che in questo caso se l'utente non salva i dati, allora dovrà successivamente specificare msb, lsb, div e dimensioni della foto da ricercare.

2.3.4. File binari

Questa modalità è la più versatile, infatti permette di nascondere sulle immagini qualunque tipologia di file purché vi sia abbastanza spazio sulla foto stessa. Addirittura, è possibile conservare delle intere directory.

Il funzionamento è analogo a quello della modalità per i file testuali, le differenze principali però riguardano il fatto che il programma legge e scrive in modalità di lettura e scrittura binaria. Chiede anche all'utente se prima di occultare i dati egli desidera comprimere questi ultimi (per poter ottimizzare lo spazio offerto dalle immagini) e permette a lui di conservare anche delle directory (che in questo caso verranno sempre salvate come una cartella in uno .zip). Se l'utente decide di comprimere i dati, il programma si occuperà di tutto (compressione, occultamento/recupero, eliminazione di file temporanei...).

La procedura per il resto è identica a quella per i file testuali.

Anche i parametri utilizzati per nascondere i dati sono gli stessi (n e div), ma in questo caso, se l'utente volesse recuperare i dati senza fornire un file di recupero o senza utilizzare i dati salvati in automatico dal programma (dati recenti), allora dovrà specificare anche se i dati che vuole recuperare sono stati compressi e a quanto ammonta la quantità di dati da estrarre (nonché ricordare la tipologia di file nascosto).

3. Descrizione tecnica del Tool

3.1. Funzioni di avvio e interfaccia

All'avvio il programma si occupa di mostrare la schermata del titolo e successivamente il menu principale:

```
1505  def start() -> None:
1506      system("cls")
1507      print("Con cosa vuoi operare?")
1508      print("1. Stringhe di testo")
1509      print("2. File a caratteri")
1510      print("3. Immagini dentro immagini")
1511      print("4. File binari")
1512      print("5. \33[31mESCI\33[0m")
1513      ans = input("--> ")
1514      while ans == "" or not str(ans).isdigit() or int(ans) not in [1,2,3,4,5]:
1515          print("\33[1;31mERRORE\33[0m: inserisci un valore valido")
1516          system("pause")
1517          system("cls")
1518          print("Con cosa vuoi operare?")
1519          print("1. Stringhe di testo")
1520          print("2. File a caratteri")
1521          print("3. Immagini dentro immagini")
1522          print("4. File binari")
1523          print("5. \33[31mESCI\33[0m")
1524          ans = input("--> ")
1525      ret = mode(int(ans))
1526      if ret:
1527          start()
1528
1529  def ascii_art() -> None:
1530      colors = ["red","yellow","green","cyan","blue","magenta"]
1531      str = "STEGANOGRAFIA\nversione 2.077"
1532      print(colored(figlet_format(str, font="slant"), colors[randint(0,5)], 'on_black', ['bold', 'blink']), end='')
1533      print("by ", end='')
1534      arcobaleno("Giuseppe Bellamacina")
1535      print("\n")
1536      system("pause")
1537
1538  def main():
1539      system("cls")
1540      colorama.init()
1541      ascii_art()
1542      start()
1543
1544  main()
```

Di cruciale importanza è la funzione mode() che si occupa di avviare le funzioni corrette dopo che l'utente seleziona la modalità desiderata (di sotto una porzione di essa):

```
1332  def mode(mod: int) -> bool:
1333      global n_backup, div_backup, lsb_backup, msb_backup, size_backup, zipMode_backup
1334      global w_backup, h_backup, img_with_data_backup, img_with_data_name_backup, mode_backup
1335      system("cls")
1336
1337      # hideMessage()
1338      if mod == 1:
1339          sub = subMode()
1340          if sub == 1:
1341              img = imgInput()
1342              msg = msginput()
1343              new_img = imgOutput()
1344              img_with_data_backup = hideMessage(img, msg, new_img)
1345              img_with_data_name_backup = new_img
1346              mode_backup = 1
```

3.2. Occultamento di stringhe

La funzione di occultamento delle stringhe verifica innanzitutto se la foto selezionata può contenere la stringa inserita e la converte in formato RGB nel caso in cui non lo fosse prima. Dopo effettua una copia dell'immagine di input e lavora sulla copia. Prende l'intera stringa e la converte in una stringa binaria a cui aggiunge la stringa "00000000" formata da 8 zeri consecutivi che fungerà da carattere terminatore per la funzione di estrazione della stringa.

La funzione legge l'immagine trattandola come una matrice di pixels e setta su ogni canale dei primi pixels necessari a nascondere l'intero messaggio, l'ultimo bit meno significativo ad uno di quelli da celare. Alla fine della procedura ritorna la copia modificata.

```python
def hideMessage(img: Image, msg: str, new_img: str) -> Image:
    """Nasconde una stringa in una foto"""
    system('cls')
    # controlla se l'immagine è abbastanza grande
    if (img.width * img.height) * 3 < len(msg) * 8:
        f = img.filename.split("\\")[-1]
        print(f"\33[1;31mERRORE\33[0m: Immagine \33[31m{f}\33[0m troppo piccola per nascondere il messaggio")
        system("pause")
        return img
    # converte in RGB
    if img.mode != "RGB":
        img = img.convert("RGB")
    # inizia a nascondere
    arcobaleno("OCCULTAMENTO MESSAGGIO")
    print("...")
    img_copy = img.copy()
    mat = img_copy.load()
    msg = binaryConvert(msg)
    msg = msg + "00000000"
    msg = list(msg)
    for i in range(img.width):
        for j in range(img.height):
            for z in range(3):
                if msg != []:
                    bit = msg.pop(0)
                    color = mat[i,j][z] # ottieni il colore
                    color = setLastBit(color, bit) # cambia l'ultimo bit
                    mat = setComponentOfColor(mat, i, j, color, z) # setta il colore
                else:
                    break
    print(f"\33[1;32mTERMINATO\33[0m\nPercentuale di pixel usati: {format(((len(msg) / ((img.width * img.height) * 3)) * 100), '.2f')}%")
    print(f"Immagine salvata come \33[33m{new_img}\33[0m")
    img_copy.save(new_img)
    return img_copy
```

La funzione di recupero fa essenzialmente la stessa cosa, ma al contrario.

```python
def getMessage(img: Image) -> str:
    """Ottieni un messaggio nascosto"""
    system('cls')
    if img.mode != "RGB":
        img = img.convert("RGB")
    # inizia la procedura
    mat = img.load()
    msg, stop = [], []
    for i in range(img.width):
        for j in range(img.height):
            for z in range(3):
                color = mat[i,j][z]
                bit = format(color, '08b')[-1]
                stop.append(bit)
                msg.append(bit)
                if len(stop) == 8:
                    if ''.join(stop) == "00000000":
                        msg = ''.join(msg)
                        msg = msg[:-8]
                        msg = binaryConvertBack(msg)
                        return msg
                    stop = []
    msg = ''.join(msg)
    msg = msg[:-8]
    msg = binaryConvertBack(msg)
    return msg
```

3.3. Occultamento di file testuali

La funzione controlla inizialmente se n è valido se è inserito manualmente, altrimenti lo calcola in maniera automatica. Successivamente controlla se l'immagine può contenere i dati che si vogliono inserire. Se tutto va bene trasforma l'immagine in RGB e inizia a scorrerla con un array lineare composto da tutte le componenti ordinate di tutti i pixels. Era presente anche una versione che scansionava la foto come una matrice a 3 strati (RGB), ma è deprecata poiché la versione con l'array lineare è più veloce. La funzione calcola anche un div automatico se non è stato specificato.

```
255  def findDiv(dim: int, file: str, n: int) -> float: # più stabile
256      image_dim = dim * n
257      div = ((image_dim - n) / (getsize(file) * 8))
258      return div
```

Per il resto la funzione opera come quella per le stringhe, la differenza principale riguarda il fatto che essa si itera per ogni riga del file da inserire.

```
296      start_time = time.time()
297      with open(file, 'r', encoding='utf-8') as f:
298          total_lines = sum(1 for line in f)
299          f.seek(0)
300          rsv = ""
301          ind, pos = 0, 0
302          for i, line in enumerate(f):
303              line = binaryConvert(line)
304              line = rsv + line
305              rsv = ""
306              if i == total_lines - 1:
307                  line = line + "00000000"
308              line = list(line)
309              flag = False
310              while line != []:
311                  bit = ""
312                  for k in range(n):
313                      if len(line) >= n-k:
314                          bit += line.pop(0)
315                      else:
316                          line = ''.join(line)
317                          rsv += line
318                          flag = True
319                          break
320                  if flag:
321                      break
322                  arr[pos] = setLastNBits(arr[pos], bit, n)
323                  ind += div
324                  pos = round(ind)
325              if (i + 1) % 3000 == 0:
326                  system('cls')
327                  arcobaleno("OCCULTAMENTO FILE")
328                  print("...")
329                  progress = (i + 1) / total_lines
330                  elapsed_time = time.time() - start_time
331                  print(f"Elaborate {i + 1} righe su {total_lines} ({format((progress) * 100, '.2f')}%)")
332                  print(f"Tempo rimanente: {timeDisplay(elapsed_time *(1 - progress) / progress)}")
```

Al termine della prima fase, la funzione controlla se ci sono bit nel buffer utilizzato per settare i bit meno significativi della foto a partire da quelli estratti dal file. In questo caso li inserisce e torna un messaggio all'utente informandolo dell'avvenuta procedura. Per estrarre i dati la funzione è la medesima ma ovviamente funziona al contrario. In quel caso si crea un file.

3.4. Occultamento di immagini

La funzione controlla se i parametri lsb e msb sono validi e compatibili, vale a dire se appartengono all'intervallo tra 0 e 8 e se lsb è minore di msb. Altrimenti dà un errore oppure se richiesto li calcola in maniera dinamica. Dopo verifica che sia tutto corretto e se le due immagini possono essere nascoste una nell'altra. A questo punto converte le immagini in formato RGB e le serializza in un array lineare. Vengono scansionate in questo modo e così facendo si prendono le info necessarie della seconda, si inseriscono in un buffer e questi ultimi si immettono nella prima.

```
    pos = 0
    bit_queue = ""
    while i < len(arr2):
        # extract msb bits from each channel of arr2
        r, g, b = arr2[i], arr2[i + 1], arr2[i + 2]
        bit_queue += format(r, '08b')[:msb] + format(g, '08b')[:msb] + format(b, '08b')[:msb]
        while len(bit_queue) >= lsb * 3:
            # extract lsb bits from each channel of arr1
            r_bits, g_bits, b_bits = bit_queue[:lsb], bit_queue[lsb:2*lsb], bit_queue[2*lsb:3*lsb]
            arr1[j] = setLastNBits(arr1[j], r_bits, lsb)
            arr1[j + 1] = setLastNBits(arr1[j + 1], g_bits, lsb)
            arr1[j + 2] = setLastNBits(arr1[j + 2], b_bits, lsb)
            # remove lsb bits from bit_queue
            bit_queue = bit_queue[lsb*3:]
            pos += 3 * div
            j = round(pos)
        i += 3
        if i % 180000 == 0:  # Change this number to control how often the progress is printed
            system("cls")
            arcobaleno("OCCULTAMENTO IMMAGINE")
            print("...")
            progress = i / total_pixels_ch
            elapsed_time = time.time() - start_time
            print(f"Elaborazione in corso: {format(progress * 100, '.2f')}%")
            print(f"Tempo rimanente: {timeDisplay(elapsed_time * (1 - progress) / progress)}")
    # convert arr1 to image
    w, h = img2.width, img2.height
```

Si noti che per inserire le informazioni nella prima foto si utilizza la seguente funzione:

```
def setLastNBits(value: int, bits: str, n: int) -> int:
    """Setta gli ultimi n bits di un numero"""
    value = format(value, '08b')
    if len(bits) < n:
        n = len(bits)
    value = value[:-n] + bits
    value = int(value, 2)
    value = min(255, max(0, value))
    return value
```

Procedura analoga anche per la funzione di estrazione, ma in questo caso si deve creare un'immagine vuota da riempire gradualmente con i dati estratti dall'immagine data in input. Ciò si può fare creando un array vuoto e convertendolo in formato immagine alla fine.

```
    start_time = time.time()
    size = width * height * 3
    arr = np.array(img).flatten().copy()
    res = np.zeros(size , dtype = np.uint8)
    bits = ""
    pos, j, n = 0, 0, 0
    while n < size:
        # get the lsb less significant bits of each pixel of img1
        bits += format(arr[j], '08b')[-lsb:] # remove the lsb less significant bits
        if len(bits) >= msb:
            tmp = bits[:msb]
            # add 0s to tmp to get a byte
            while len(tmp) < 8:
                tmp += "0"
            # get the msb most significant bits of each pixel of img2
            res[n] = int(tmp, 2)
            n += 1
            bits = bits[msb:]
        pos += div
        j = round(pos)
        if (n+1) % 180000 == 0:
            system("cls")
            arcobaleno("RICERCA IMMAGINE")
            print("...")
            progress = n / size
            print(f"Elaborazione in corso: {format(progress * 100, '.2f')}%")
            elapsed_time = time.time() - start_time
            print(f"Tempo rimanente: {timeDisplay(elapsed_time * (1 - progress) / progress)}")
    system("cls")
    arcobaleno("RICERCA IMMAGINE")
    print("...")
    print("Elaborazione in corso: 100.0%")
    print("Tempo rimanente: 0 secondi")
    # convert res to image
    res = Image.fromarray(res.reshape(height, width, 3))
```

3.5. Occultamento di file binari e cartelle

Il funzionamento è identico a quello della funzione per i file testuali, ma qui si leggono i file in modalità binaria e non è presente una stringa di terminazione.

```
    with open(file, 'rb') as f:
        f.seek(0)
        for i in range(total_bytes):
            byte = f.read(1) # read byte
            bits = format(ord(byte), '08b') # convert byte into string of bits
            bits = rsv + bits
            rsv = ""
            while len(bits) >= n:
                tmp = bits[:n]
                bits = bits[n:]
                # set last n bits of pixel
                arr[pos] = setLastNBits(arr[pos], tmp, n)
                ind += div
                pos = round(ind)
            if len(bits) > 0:
                rsv = bits
```

Anche il processo inverso è analogo. Un'altra differenza è che se si vogliono comprimere i file prima di avviare la procedura (compressione obbligatoria per occultare directory), il programma si occuperà di creare un file temporaneo .zip.

3.6. Dati di backup

Il programma gestisce i dati di recupero delle informazioni di recupero in maniera automatica se si tratta di recuperare le informazioni dall'ultima procedura di occultamento, mentre richiede un file di recupero o di inserire i dati manualmente per tutte le altre.

```
1247  def saveData()-> None:
1248      global n_backup, div_backup, lsb_backup, msb_backup, size_backup, zipMode_backup
1249      global w_backup, h_backup, img_with_data_name_backup, mode_backup
1250      system("cls")
1251      print("Vuoi salvare i dati? (Y/N)")
1252      ans = input("--> ")
1253      ans = ans.upper()
1254      while ans == "" or (ans != "Y" and ans != "N"):
1255          print("\33[1;31mERRORE\33[0m: inserisci un valore valido")
1256          system("pause")
1257          system("cls")
1258          print("Vuoi salvare i dati? (Y/N)")
1259          ans = input("--> ")
1260          ans = ans.upper()
1261      if ans == "Y":
1262          while True:
1263              system("cls")
1264              print("Inserisci il nome del file di output")
1265              out = input("File --> ")
1266              try:
1267                  f = open(out, 'w')
1268                  f.close()
1269                  break
1270              except:
1271                  print("\33[1;31mERRORE\33[0m: file non creato correttamente")
1272                  system("pause")
1273          backup = [n_backup, div_backup, lsb_backup, msb_backup, size_backup, w_backup, h_backup, img_with_data_name_backup, zipMode_backup, mode_backup]
1274          with open(out, "wb") as file:
1275              pickle.dump(backup, file)
1276          system("cls")
1277          print("\33[1;32mDATI SALVATI\33[0m")
1278          system("pause")
```

4. Conclusioni

Il programma funziona correttamente in tutte le sue modalità, non presenta bug evidenti di alcun tipo ed è molto versatile grazie alla sua meccanica di personalizzazione delle procedure tramite parametri.

Si possono conservare grandi quantità di dati su immagini di risoluzione discreta senza comprometterne eccessivamente la qualità ed è possibile estrarre i dati con facilità.

I processi più articolati come quello delle immagini nelle immagini o quello dei file binari sono molto resistenti anche contro tool di Steganalisi poiché riescono a nascondere i dati in maniera non evidente (specie se si utilizzano le modalità automatiche).

I tempi di esecuzione di tutte le procedure sono ottimizzati grazie ai processi di serializzazione dei dati delle immagini.

Giuseppe Cosimo Alfio Bellamacina

1000030349