Departamentul Automatică şi Informatică Industrială
Facultatea Automatică şi Calculatoare
Universitatea POLITEHNICA din Bucureşti
Splaiul Independenţei 313, 060042, Bucureşti, România
Sala ED 412, Tel. 021/402.92.69
www.aii.pub.ro, email: secretariat@aii.upb.ro

Sesiunea de Comunicări Şiinţifice Studenţeşti

Ediţia 2022

Steganosaurus Crypt

-Protecţie combinată bazată pe steganografie şi criptare-

Autor: Matei Mihnea-Cristian, Facultatea de Automatica, Calculatoare si Ingineria Sistemelor, Anul III, grupa 331AB

Adresa e-mail: matei.mihnea574@gmail.com

Îndrumător ştiinţific: Conf.dr.ing. Ştefan-Alexandru MOCANU

Cuprins

1. Introducere

Steganosaurus Crypt este o aplicație ce are ca scop ascunderea de mesaje secrete sau imagini în interiorul altor imagini. În contextul unei lumi în care siguranța și securitatea datelor sunt din ce în ce mai greu de asigurat, prezenta aplicație oferă un grad suplimentar de protecție asupra unor date pe care utilizatorul le vrea în siguranță. La baza aplicației se află un algoritm simplu, ce folosește biții cei mai puțin semnificativi din canalele pixelilor unei imagini, loc în care se ascunde mesajul, criptat sau nu. Algoritmul de criptare al mesajelor este reprezentat de efectuarea unei operații SAU-EXCLUSIV între o cheie generată și mesajul ce va fi ascuns. Dat fiind scopul acestei unelte de ascundere a datelor, domeniile pe care aceasta le vizează sunt steganografia [11] și criptografia [12].

Steganografia (din lb. gr. Steganos-ascuns & graphia-scris) este o practică, folosită încă din timpuri străvechi, ce are ca scop ascunderea unui mesaj în interiorul unui alt mesaj sau obiect. Conform scrierilor lui Herodot [10], prima apariție a ceea ce numim acum steganografie datează din sec. V, înainte de Hristos. Conducătorul Miletului, tiranul grec Histiaeus, tundea părul sclavilor, după care le tatua pe scalp mesaje despre mișcări ale armatei persane. Acesta aștepta că părul indivizilor să crească înapoi, după care îi trimitea destinatarului. Avantajul principal al steganografiei este că obiectul purtător nu semnalează prezența unui mesaj ascuns. Astfel, informația ascunsă nu atrage atenția asupra sa și va trece, de cele mai multe ori, neobservată.

Steganografia digitală se face, de obicei, prin intermediul nivelului de transport: documente, imagini, programe sau protocoale. Ideale sunt fișierele media, întrucât acestea au o dimensiune mare. Spre exemplu, în cazul unei imagini, ajustarea biților cei mai puțin semnificativi în așa fel încât aceștia să formeze un mesaj, produce o schimbare insesizabil ochiului uman.

Criptografia (din lb. gr. kryptos-secret & graphia-scris), sau criptologia, este practica și studiul tehnicilor ce au ca scop securizarea comunicațiilor. În general, criptografia se referă la construirea și analiza unor protocoale ce previn citirea de către părți terțe a mesajelor și informațiilor private. Aceasta se bazează pe matematică, fizică, informatică și inginerie electrică și este folosită pe mai multe arii de interes, precum: comerțul electronic, parole, comunicații militare. Mesajul este criptat astfel încât dintr-o informație inteligibila, prin diferite transformări, să devină de neînțeles pentru o parte terță ce nu se află în posesia mecanismului de decriptare.

Criptarile de date se fac în baza unor algoritmi matematici ce folosesc chei de criptare (mesajul este transformat în funcție de acestea). Astfel, cunoașterea algoritmului nu prezintă un pericol (de multe ori acesta este bine-cunoscut), cât timp cheia rămâne doar în posesia destinatarului.

2. Descrierea problemei și prezentarea soluției propuse

Steganosaurus Crypt este o aplicație ce are ca scop ascunderea unui mesaj sau a unei imagini în interiorul altei imagini fără a atrage atenția asupra acestui fapt. Pentru un grad mai înalt de protecție, înainte ca mesajul să fie ascuns în imagine, acesta va fi criptat cu ajutorul unei chei generate aleator. Problema este reprezentată de modalitatea de ascundere a informației, dar și de interacțiunea cu utilizatorul, care trebuie să fie cât mai facilă. Putem presupune următorul scenariu: Joe vrea să îi trimită prietenului său Vlad informații cu privire la locurile unde fratele său mai mare, Vladimir, își ascunde jucăriile. Problema este că toate mesajele primite de Vladut trec prin e-mail-ul lui Vladimir. Cum ar putea Joe să îi transmită informația lui Vlad, fără ca Vladimir să afle?

Soluția propusă constă într-un program care îi permite lui Joe să mascheze informațiile pe care dorește să i le transmită lui Vlad într-o imagine. În aparență, imaginea nu dă de gol prezența unui mesaj, acesta putând fi găsit din nou prin intermediul aceluiași program. Astfel, soluția propusă nu oferă utilizatorului doar posibilitatea de a masca mesajul, ci oferă și posibilitatea unui utilizator-destinatar de a căuta un mesaj ascuns. În momentul ascunderii mesajului, acesta va fi criptat, adăugând un nivel în plus de siguranță, în cazul în care Vladimir bănuiește totuși ceva.

La baza soluției stă steganografierea folosind tehnica bit-ului cel mai puțin semnificativ (în eng. Least Significant Bit), după cum este descrisă în [7]. O imagine digitală poate fi descrisă ca un set finit de valori numite pixeli. Pixelii sunt cel mai mic element ce conține informație al unei imagini, conținând valorile care reprezintă culoarea acesteia într-un anumit punct. Practic, imaginea este o matrice de pixeli, ce are un număr fix de coloane și rânduri. Fiecare pixel este format din 3 canale de culoare: roșu, verde și albastru. Pentru steganografierea unui mesaj,

tehnica LSB implementată înlocuiește ultimul bit al fiecărui canal din fiecare pixel cu câte un bit al mesajului secret. Schimbarea valorii pixelului respectiv este cu +/-3, așadar o schimbare de aproximativ 0.000002%, complet insesizabilă ochiului uman, deci un impact minim asupra rezultatului. În cazul în care mesajul secret este reprezentat de o imagine, este folosită aceeași tehnică, însă vor fi înlocuiți ultimii 3 sau 4 biți ai imaginii purtătoare cu primii 3 sau 4 biți ai imaginii ce trebuie ascunsă. Având în vedere principiul de bază al steganografiei (imaginea purtătoare nu trebuie să dea de bănuit că ascunde ceva), pentru ca informația secretă să fie mai bine ascunsă, cazul în care sunt ascunși primii 3 biți este de preferat. Dezavantajul pe care această variantă îl prezintă este calitatea mult redusă a imaginii recuperate față de imaginea originală. Pe de altă parte, varianta de păstrare a 4 biți recuperează o imagine vizibil mai calitativă decât cea recuperată de varianta prezentată anterior.

Pentru a se asigura că imaginea nu prezintă niciun indiciu pentru Vladimir, Joe ar trebui să folosească o imagine proprie, care nu se găsește pe internet (o imagine ‘capturată’ ar putea fi găsită pe internet și comparată cu aceasta) și care prezintă o omogenitate scăzută (nu se găsesc zone mari de pixeli ce conțin aceleași valori).

În acest mod, Joe poate transmite prietenului său orice informație dorește despre locurile în care Vladimir își ascunde mult iubitele jucării fară a-și face griji că acesta va descoperi conversația dintre ei.

3. Prezentarea aplicației

3. 1. Interfața grafică

Steganosaurus Crypt este o aplicație ce permite utilizatorului să ascundă un mesaj criptat sau o imagine într-o altă imagine numită purtător. Imaginea purtător nu trebuie să ofere niciun indiciu cu privire la conținutul secret pe care îl poartă. Scopul este protejarea unor date confidențiale, pe care utilizatorul să le poată transmite unui destinatar. Destinatarul poate, prin intermediul aceleiași aplicații, să recupereze informația secretă din imaginea purtătoare.

Aplicația prezintă o interfață grafică prietenoasă, intuitivă, ce a fost construită prin intermediul librăriei Tkinter [9]. Astfel, utilizatorul poate folosi cu ușurință aplicația și se poate folosi ușor de uneltele acesteia.

Fig. 1. Meniul Principal al aplicației

După cum se poate observa și în Fig. 1., meniul principal al aplicației prezintă informații esențiale cu privire la locația fișierelor de intrare și de ieșire, dar și două butoane care conduc utilizatorul spre meniurile de ascundere a informației (“Encode”) și de recuperare (“Decode”). În interiorul acestor meniuri, utilizatorului îi este oferită opțiunea de a alege tipul de date pe care dorește să îl ascundă sau să îl recupereze, precum și informații cu privire la datele ce urmează a fi introduse, după cum poate fi observat în Fig. 2., după care, în urma selecției, îi va fi deschisă fereastra aferentă selecției făcute. În Fig. 3. poate fi observat codul din spatele ferestrei aferente meniului de ascundere (“Encode”). Fiecare element al ferestrei este așezat manual în aceasta, în funcție de o valoare procentuală a lungimii și înălțimii ferestrei (similar pentru toate ferestrele). Ferestrele fiecărei opțiuni conțin câmpuri în care utilizatorul va introduce datele cerute (numele fișierelor de intrare și ieșire, informația – pentru ascundere, alte opțiuni suplimentare), un buton care efectuează operația cerută, în funcție de datele introduse și un buton pentru revenirea la meniul anterior (vezi Fig. 4. și Fig. 5.). În momentul apăsării butonului de execuție, o secvență de funcții specifică butonului respectiv. Funcțiile vor fi detaliate individual în paragrafele ce urmează.

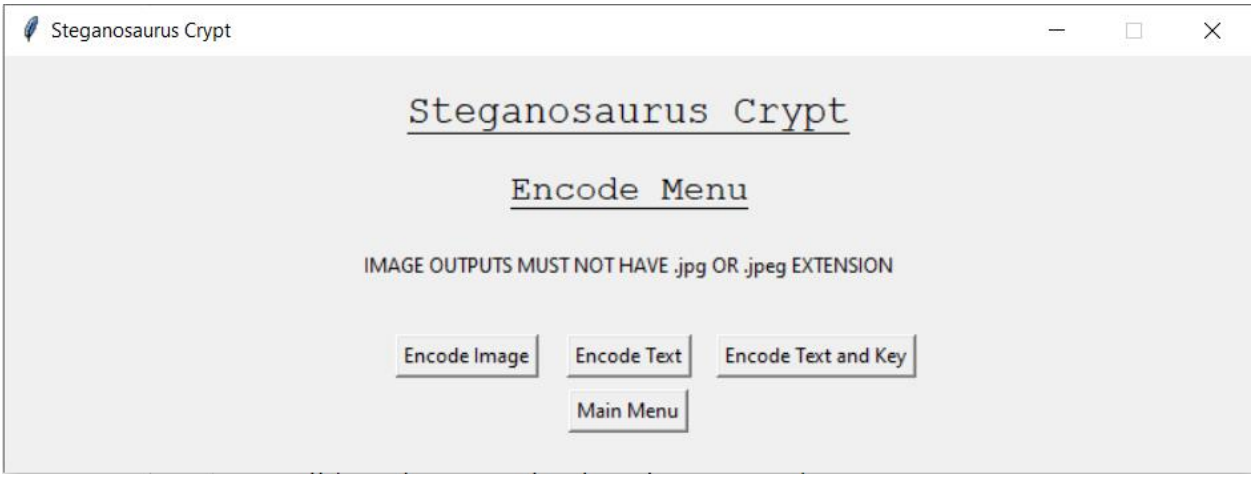

Fig. 2. Meniul de Encode (similar meniului Decode)

```
215 def window_encode():
216     encode_window = Toplevel(win)
217     encode_window.geometry("750x250")
218     encode_window.title("Steganosaurus Crypt")
219     encode_window.resizable(width=False, height=False)
220 
221     label_mess2=Label(encode_window, text="Steganosaurus Crypt", font= ('Courier 18 underline'))
222     label_mess2.place(relx=0.5, rely=0.13, anchor=CENTER)
223     label_mess=Label(encode_window, text="Encode Menu", font= ('Courier 16 underline'))
224     label_mess.place(relx=0.5, rely=0.32, anchor=CENTER)
225 
226     btn_encodeimg = Button(encode_window, text="Encode Image", command=lambda:[encode_window.withdraw(), win_enc_img3(encode_window)])
227     btn_encodeimg.place(relx=0.37, rely=0.72, anchor=CENTER)
228 
229     btn_encodemes = tk.Button(encode_window, text="Encode Text", command=lambda:[encode_window.withdraw(), win_enc_txt(encode_window)])
230     btn_encodemes.place(relx=0.5, rely=0.72, anchor=CENTER)
231 
232     btn_encodemes = tk.Button(encode_window, text="Encode Text and Key", command=lambda:[encode_window.withdraw(), win_enc_header(encode_window)])
233     btn_encodemes.place(relx=0.65, rely=0.72, anchor=CENTER)
234 
235     btn_mm = tk.Button(encode_window, text="Main Menu", command=lambda:[encode_window.withdraw(), win.deiconify()])
236     btn_mm.place(relx=0.5, rely=0.85, anchor=CENTER)
237 
238     note2 = Label(encode_window, text="IMAGE OUTPUTS MUST NOT HAVE .jpg OR .jpeg EXTENSION")
239     note2.place(relx=0.5, rely=0.5, anchor=CENTER)
240 
```

Fig. 3. Codul din spatele ferestrei meniului de Encode

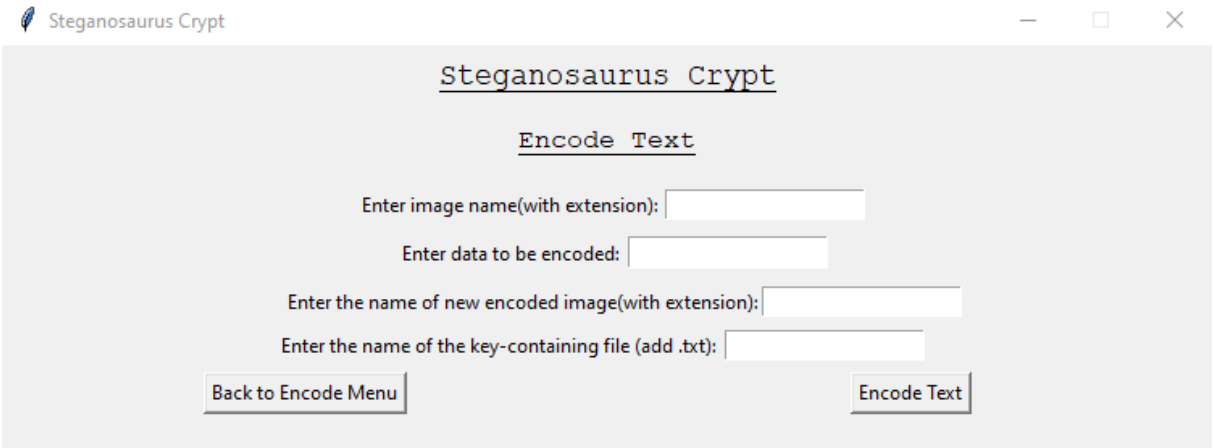

Fig. 4. Fereastra opțiunii de ascundere text cu cheie în fișier

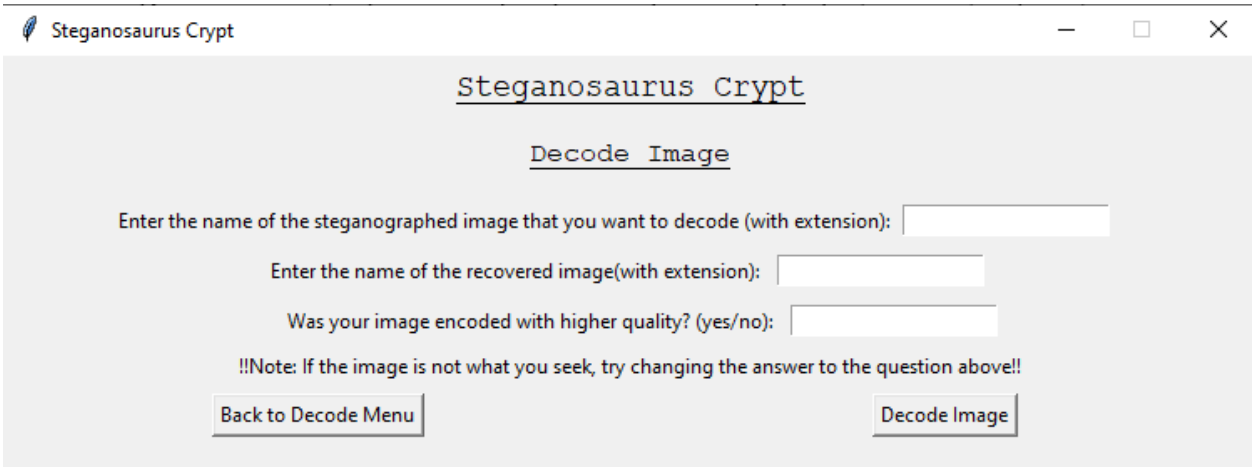

Fig. 5. Fereastra opțiunii de recuperare imagine ascunsă

3.2. Meniul "Encode"

Meniul "Encode" prezintă utilizatorului posibilitatea de a ascunde într-o imagine purtătoare două tipuri de date: imagine sau text. În cazul ascunderii unei informații de tip text, aceasta va fi criptată mai întâi, apoi ascunsă, fiind oferită posibilitatea de integrare a cheii de decriptare în header-ul aceleiași imagini purtătoare sau într-un fișier text aflat în directorul aplicației.

3.2.1. Ascunderea unei imagini într-o altă imagine – "Encode Image"

Funcția encode_img(param1, param2, param3, param4), în care parametrii 1 și 2 sunt reprezentați de numele fișierelor de intrare, parametrul 3 este numele fiserului de ieșire, iar al 4-lea parametru este calitatea imaginii ascunse (înaltă – 4 biți ascunși, joasă – 3 biți ascunși). În interiorul funcției, imaginile sunt deschise cu ajutorul librăriei OpenCV-Python [13] și sunt comparate dimensiunile imaginii purtătoare cu cele ale imaginii secrete: în cazul în care acestea diferă, imaginea ce va fi ascunsă vă fi redimensionată la dimensiunile purtătoarei. În continuare, va fi apelată funcția care efectuează operația de ascundere propriu-zisă. În cazul în care datele introduse nu sunt valide, aplicația va da eroare și se va opri, fiind necesară repornirea. Dacă datele sunt în regulă, la finalul acestei funcții va fi scrisa imaginea în memorie cu ajutorul aceleiași librării.

Funcția hideData(param1, param2, param3): parametrii 1, respectiv 2 reprezintă imaginea purtătoare, respectiv imaginea ce urmează a fi steganografiata, iar parametrul 3 este numărul de biți ai imaginii ascunse, ce vor înlocui ultimii biți ai imaginii purtătoare. Imaginile sunt parcurse simultan, pixel cu pixel, fiind accesate canalele R, G și B ale fiecărui pixel din ambele imagini. Valorile canalelor respective sunt extrase și convertite în binar cu ajutorul unei funcții de conversie [7], după care o valoare nouă, aferentă fiecărui canal, este formată din primii 8-n biți ai imaginii purtătoare și primii n biți ai imaginii ascunse, unde n este numărul de biți ce va fi ascuns în funcție de calitatea dorită.

3.2.2. Ascunderea unui text într-o imagine cu cheie în fișier – "Encode Text"

Funcția encode_text(param1, param2, param3, param4) cuprinde 2 parametrii cu date de intrare (param1 – imaginea purtătoare & param2 – informația ascunsă) și 2 parametrii cu date de ieșire (param3 – numele final al purtătoarei & param4 – numele fișierului ce conține cheia de criptare). Este verificat dacă a fost introdusă informație în parametrul 2, după care este apelată funcția hideDataMessage. La finalul acestei funcții, noua imagine vă fi scrisă în memorie.

Funcția hideDataMessage(param1, param2, param3, param4), unde primul parametru este imaginea purtătoare, al doilea este reprezentat de mesajul secret, al treilea este un parametru de decizie (dacă se dorește integrarea cheii de criptare în header-ul purtătoarei sau nu) și al patrulea este numele fișierului în care va fi găsită cheia (pentru opțiunea aferentă). Este verificată lungimea mesajului (dacă acesta nu poate fi introdus, utilizatorului îi va fi prezentat un mesaj de eroare corespunzător) și este generată și scrisă în fișier cheia de lungimea mesajului introdus. Mesajul secret va fi convertit în cod binar și va fi criptat cu o criptare de tip XOR, prin intermediul funcției MEXOR(cheie, mesaj, parametru_selectie) (Fig. 6.) După criptare, mesajului îi este adăugat un terminator și este introdus bit cu bit în imaginea purtătoare (Fig. 7.), procedeu asemănător celui descris la funcția hideData: imaginea este parcursă pixel cu pixel, sunt extrase canalele pixelilor și este introdus mesajul bit cu bit în ultimul bit al fiecărui canal, până când, se atinge lungimea mesajului cu terminator.

```
12
13  #function that uses the XOR operation between key and message so the message gets encrypted
14  def MEXOR(key, secret_message, n):
15      if (n == 0):
16          key = messageToBinary(key)
17      else:
18          key = [ key[i: i+8] for i in range(0, len(key), 8) ]
19      secret_message = messageToBinary(secret_message)
20      secret_message = [ secret_message[i: i+8] for i in range(0, len(secret_message), 8) ]
21      xor_mess = []
22      for key_val, mess_val in zip(key, secret_message):
23          xor_mess.append(int(key_val,2) ^ int(mess_val,2))
24      final_message = ""
25      for val in xor_mess:
26          val = messageToBinary(val)
27          final_message += val
28      return final_message
29
```

Fig. 6. Codul funcției MEXOR

```
43      secret_message = MEXOR(key, secret_message, 0)
44      terminator = "#"
45      terminator = messageToBinary(terminator)
46      secret_message += terminator # message terminator
47      data_index = 0
48      binary_secret_msg = secret_message
49      data_len = len(binary_secret_msg) #Find the length of data that needs to be hidden
50      for values in image:
51          for pixel in values:
52            # convert RGB values to binary format
53              r, g, b = messageToBinary(pixel)
54            # modify the least significant bit only if there is still data to store
```

Fig. 7. Secțiune din codul funcției hideDataMessage

3.2.3. Ascunderea unui text într-o imagine și a cheii în header – “Encode Text and Key”

Funcția encode_text_key(param1, param2, param3) este aproape identică cu funcția encode_text, lipsind parametrul 4 al acesteia, întrucât cheia nu va mai fi înscrisă într-un fișier. Este apelată funcția hideDataMessage, însă la finalul acestei funcții imaginea nu va mai fi scrisă în memorie cu ajutorul librăriei OpenCV, ci va fi executată o secvență de cod ce creează binar un fișier de tip .png. Codul executat este asemănător cu cel descris la [8]. Funcția apelată , primește numele purtătoarei, fișierul de ieșire și cheia și apelează mai departe funcția dump_png(fisier_iesire, purtătoare, cheie) care face scrierea efectivă a chunk-urilor prezente într-o imagine.

Funcția dump_png(fisier_iesire, purtătoare, cheie) are ca scop scrierea binară a unei imagini de tip .png în memorie conform structurii descrise la [1]. Mai întâi este scrisă semnătura tipului de fișier, o secvență de 8 bytes care conține 8 valori numerice găsite la începutul oricărui

fișier de tip .png. Sunt extrase și initializate informații cu privire la imaginea purtătoare, informații ce vor fi înscrise în chunk-ul IHDR al header-ului (chunk critic ce conține informații esențiale cu privire la imaginea ce urmează a fi procesată). Chunk-ul IHDR este creat folosind o funcție numite make_ihdr, ce are ca parametrii informațiile esențiale ce ar trebui să se afle în acesta. Această funcție returnează un obiect format din bytes, formatate specific tipului de fișier. Datele compresate sunt trimise funcției chunk(fisier_iesire, nume_chunk_binar, data). Similar este creat și chunk-ul IDAT, care conține imaginea propriu-zisă, imaginea purtătoare (care conține mesajul) fiind compresata cu ajutorul funcțiilor make_idat și compress_data și transmisă la rândul ei funcției chunk. Cu ajutorul unei funcții encode_data, înainte de înscrierea în fișier a imaginii, vor fi inversate canalele R și B pentru a se trece la modelul RGB utilizat de Windows. În continuare, chunk-ul ce va conține cheia simetrică a mesajului nostru criptat va fi ascunsă într-un chunk de tip non-critic, numit puNK. Acesta va fi creat și înscris în fișier în aceeași maniera ca și chunk-urile precedente. Ultimul chunk înscris va fi IEND, care nu conține informație, însă este critic fișierului.

Este important de știut că ordinea în care aceasta funcție chunk este apelată este extrem de importantă, întrucât un fișier .png trebuie obligatoriu să înceapă cu chunk-ul IHDR, să conțină unul sau mai multe chunk-uri IDAT și un chunk final IEND.

```
49 def dump_png(out, img, key):
50     out.write(HEADER)  # start by writing the header
51     assert len(img) > 0  # assume we were not given empty image data
52     width = len(img[0])
53     height = len(img)
54     bit_depth = 8  # bits per pixel
55     color_type = 2  # pixel is RGB triple
56     ihdr_data = make_ihdr(width, height, bit_depth, color_type)
57     chunk(out, b'IHDR', ihdr_data)
58     compressed_data = make_idat(img)
59     chunk(out, b'IDAT', data=compressed_data)
60     x = ""
61     for val in key:
62         x+=str(messageToBinary(val))
63     x = bytes(x, 'utf-8')
64     chunk(out, b'puNK', data=x)
65     chunk(out, b'IEND', data=b'')
```

Fig. 8. Codul funcției dump_png

3.3. Meniul “Decode”

Meniul “Decode”, similar celui “Encode” prezintă utilizatorului opțiunile de recuperare a informației ascunse. Astfel, opțiunile prezente sunt “Decode Image” – recuperare informație tip imagine și “Decode Text” și “Find Key and Decode Text” – recuperare informație de tip mesaj cu cheie aflată în fișier text pentru “Decode Text” și cheie aflată în header pentru “Find Key and Decode Text”.

3.3.1. Recuperarea unei imagini – “Decode Image”

Pentru recuperarea unei imagini, secvență de apelare a funcțiilor începe cu decode_img. Funcția decode_img(fisier_intrare, fisier_iesire, parametru_variabil) deschide imaginea purtătoare, verifică ce număr de biți ar trebui recuperați și, după caz, apelează funcția showData sau showData_quality, două funcții care se disting doar prin numărul de biți recuperați.

Funcțiile showData(purtătoare) și showData_quality(purtătoare) parcurg pixel cu pixel imaginea purtătoare și extrag canalele de culoare, din care extrag ultimii 3 sau 4 biți și creează o nouă imagine formată din biții respectivi și completarea cu biți 0, astfel încât să formeze un byte (Fig. 9.). Imaginea recuperată va fi mai slab calitativă decât imaginea originală.

```
#Function that restores the hidden image - better quality
def showData_quality(image):
    for values in image:
        for pixel in values:
            r, g, b = messageToBinary(pixel) #convert the red,green and blue values into binary format
            pixel[0] = int(r[4:] + "0000", 2);
            pixel[1] = int(g[4:] + "0000", 2);
            pixel[2] = int(b[4:] + "0000", 2);
    return image

#function that restores the hidden image - better hiding
def showData(image):
    for values in image:
        for pixel in values:
            r, g, b = messageToBinary(pixel) #convert the red,green and blue values into binary format
            pixel[0] = int(r[3:] + "00000", 2);
            pixel[1] = int(g[3:] + "00000", 2);
            pixel[2] = int(b[3:] + "00000", 2);
    return image
```

Fig. 9. Codul funcțiilor showData și showData_quality

3.3.2. Recuperarea unui text cu cheie în fișier – “Decode Text”

În aceeași manieră, secvența de recuperare a unui mesaj text cu cheie în fișier text începe prin apelarea funcției showDataMessage(purtătoare, cale_fisier_cheie). Această funcție parcurge imaginea purtătoare pixel cu pixel și extrage din fiecare canal de culoare ultimul bit, pe care îl introduce într-o variabilă numită binary_data. Variabila este împărțită apoi în bytes (secvențe de câte 8 biți), după care este căutat byte-ul care reprezintă terminatorul introdus la ascunderea mesajului. Cu ajutorul librăriei os este deschis fișierul ce conține cheia de criptare/decriptare și este apelată funcția MEXOR pentru re-efectuarea operației XOR între mesajul criptat și cheie. Noul mesaj este împărțit din nou în bytes și parcurs byte cu byte. Fiecare byte va fi transformat într-un caracter.

3.3.3. Recuperarea unui text cu cheie în header – “Find Key and Decode Text”

În cazul opțiunii de căutare și decodificare mesaj, secvența este asemănătoare celei de recuperare a unui text cu cheia de decriptare aflată într-un fișier text, însă anterior apelării funcției showDataMessage_key(purtătoare, cheie), care este identică cu funcția showDataMessage, cu excepția cheii de criptare/decriptare care nu se mai află într-un fișier, ci este obținută anterior apelării acestei funcții prin apelarea unei secvențe de cod asemănătoare cu cea descrisă la [3].

Funcția process_chunks(nume_fisier) este o funcție ce parcurge binar un fișier de tip .png. Inițial, fișierul este transmis funcției open_png(nume_fisier) pentru ca acesta să fie deschis binar și pentru a fi validată semnătură specifică fișierului .png. În cazul în care semnătura este invalidă, va fi afișat un mesaj corespunzător. Fișierul va fi citit 4 câte 4 bytes, până este întâlnit un chunk cunoscut. În momentul întâlnirii unui chunk cunoscut, va fi apelată funcția necesară parcurgerii acestuia. Spre exemplu, când parcurgerea fișierului ajunge la chunk-ul IHDR, va fi apelată funcția process_ihdr(nume_fisier). Această funcție va procesa toate informațiile pe care acesta le conține, pentru a putea avansa în citirea fișierului. Similar,

pentru fiecare chunk există o funcție de parcurgere, inclusiv pentru chunk-ul care conține cheia căutată, care va fi returnată pentru a putea fi folosită.

```
def process_punk(fh, chunk_length):
    width = fh.read(chunk_length)
    width = width.decode('utf-8')
    key = [ width[i: i+8] for i in range(0, len(width), 8) ]
    return width

def process_chunks(filename):
    fh = open_png(filename)
    idats_found = 0
    idats_bytes = 0
    while (True):
        chlen = fh.read(4)
        if (len(chlen) == 0):
            break

        chunk_length = struct.unpack('!I',chlen)[0]

        chunk_type = ""
        for i in range(4):
            s=struct.unpack('c',fh.read(1))[0]
            chunk_type += (s.decode('UTF-8'))
# If needed, information about chunks can be printed.
# Processing of the chunks is mandatory in order to get to the needed information
        if (chunk_type == "IHDR"):
            process_ihdr(fh)
        elif(chunk_type == "puNK"):
            key = process_punk(fh, chunk_length)
        else:
            data = fh.read(chunk_length)
            crc = fh.read(4)
    return key
```

Fig. 10. Secvență de cod folosită la parcurgerea fișierului

4. Limitări și dezvoltări ulterioare

Din punct de vedere al construcției, programul prezintă numeroase limitări, câteva dintre acestea fiind prezentate mai jos. De precizat este faptul că soluția fiecărei limitări reprezintă o posibilă dezvoltare ulterioară.

Informația ascunsă în imagini salvate în format .jpg sau .jpeg este imposibil de recuperat. Această limitare este cauzată de compresia specifică acestui tip de fișier, compresie lossy [5], care implică anumite pierderi de informație care sunt esențiale recuperării. Prin utilizarea tehnicii LSB este practic imposibil de trecut peste acest impediment, însă prin utilizarea unei alte tehnici de ascundere a informației, tehnica DCT-M3, descrisă în [6] este

posibil. Această tehnică asigură un grad înalt de ascundere cu schimbări minime în cadrul imaginii purtătoare.

Proporțiile imaginii originale se pierd în cazul ascunderii unei imagini în purtătoare, dacă aceasta prezintă dimensiuni diferite. Cauza este lipsa unui terminator în cazul unei imagini cu dimensiuni mai mici decât purtătoarea. Adăugarea unui terminator, în acest caz, ar păstra proporțiile și dimensiunile imaginii originale.

Recuperarea mesajului poate da greș dacă în mesajul criptat, la împărțirea acestuia în bytes, după recuperare, apare o secvență identică terminatorului introdus la ascundere. Modificarea cantitativă a acestuia (adăugarea de mai multe simboluri că terminator) micșora drastic această probabilitate. Scoaterea terminatorului și introducerea lungimii mesajului în header ar reduce total acest risc.

Este necesară introducerea fișierelor de intrare în directorul aplicației și scrierea manuală a numelor acestora (cu extensie) în cadrul aplicației. Soluția este modificarea interfeței, astfel încât fișierele să poată fi căutate și selectate oriunde în calculatorul utilizatorului.

În cazul în care se dorește integrarea cheii de decriptare în header-ul fișierului de ieșire, acesta trebuie obligatoriu să fie de tip .png, întrucât funcțiile care creează fișierul, creează un fișier .png. Soluționarea constă în construirea mai multor ramuri de construcție a unui fișier, pentru mai multe tipuri de fișiere.

Mesajul ascuns este introdus în imagine “la rând”, făcându-l astfel mai ușor de găsit în cazul unei steganalize pe imagine. Soluția ar fi introducerea mesajului în anumiți biți, după un algoritm definit (de exemplu, introducerea în al n-lea bit sau introducerea după șirul lui Fibonacci).

Altă îmbunatățire dorită este schimbarea algoritmului de criptare, întrucât folosirea unui algoritm cu cheie simetrică, cum este XOR, reduce capabilitatile de protecție oferite de criptare în cazul ascunderii cheii în header. Practic, cine ar cunoaște codul aplicației, ar ști și unde să caute cheia, lucru care nu este dorit. Folosirea unui algoritm de criptare RSA [2] ar aduce un grad de securitate foarte ridicat al mesajului.

Încă o îmbunatățire ar însemnă introducerea unui mecanism de criptare dublă a unui mesaj. Utilizatorul care dorește să ascundă un mesaj criptat într-o purtătoare ar trimite-o destinatarului, care, la rândul lui, în funcție de lungimea mesajului, ar cripta încă o dacă mesajul criptat și ar transmite din nou purtătoarea expeditorului. Expeditorul decriptează mesajul folosind cheia lui, trimite purtătoarea din nou destinatarului, unde acesta decriptează mesajul folosind cheia proprie. Astfel, nu ar exista posibilitatea existenței pe vreun canal de comunicație a unei chei sau a unei imagini neprotejate.

Eficientizarea codului și îmbunătățirea interfeței grafice reprezintă, de asemenea, un punct de interes în dezvoltarea ulterioară a aplicației.

5. Concluzii

"Steganosaurus Crypt" este o aplicație ce permite utilizatorului să ascundă informații, fără ca fișierele care conțin aceste date să dea de bănuit. Acestuia îi este prezentată o interfață prietenoasă, care îi permite să aleagă cu ușurință operațiunea dorită. În urma selectării opțiunii dorite, prin simpla completare a unor date și prin apăsarea unui buton, aplicația execută secvența de cod necesară obținerii rezultatului dorit. De asemenea, aplicația permite și recuperarea de informație secretă din anumite fișiere.

Prin dezvoltarea acestei aplicații au fost dobândite cunoștințe în domeniul steganografiei și criptografiei. A fost îmbunătățită experiența de lucru cu limbajul Python și cu librăriile specifice acestuia, atât pe partea aplicației care nu interacționează cu utilizatorul, cât și pe partea de interfață grafică.

6. Bibliografie

[1] W3C (2003), Portable Network Graphics (PNG) Specification (Second Edition)

[2] David Naccache, Emil Simion, Adela Mihaita, Ruxandra-Florentina Olimid, Andrei-George Oprina (2011), Criptografie si Securitatea Informatiei. Aplicatii.

[3] Jake Kara (2016), PNG Structure

[4] Brian de Heus (2016), Hiding a payload in PNG files with Python

[5] Khalid Sayood (2018), Introduction to Data Compression (Fifth Edition)

[6] Abdelhamid Awad Attaby, Mona F.M. Mursi Ahmed, Abdelwahab K. Alsammak (2018), Data hiding inside JPEG images with high resistance to steganalysis using a novel technique: DCT-M3

[7] Rupali Roy (2020), Image Steganography using Python

[8] darius-code-tinkerings (2021), Generating a PNG File in Python

[9] Bijay Kumar (2021), Python Tkinter Multiple Windows Tutorial

[10] Guinness World Records (2022), First use of steganography

[11] Wikipedia (2022), Steganography

[12] Wikipedia (2022), Cryptography

[13] Conf.dr.ing. Ştefan-Alexandru MOCANU, As.drd.ing. Diana Nicoleta BAICU (2022), Suport curs si laborator Aplicatii Multimedia (W3C, 2003)