



Raport final - Steganografie

ECHIPĂ: E2

Baltariu Ionuț-Alexandru
Grupa 1305A

Bejenariu Răzvan-Marius
Grupa 1306A

https://github.com/xeno-john/AC_PIM_P

Abstract

Într-o lume în care securitatea datelor este constant incertă, dorința persoanelor de a ascunde informații confidențiale sau cu mare importanță în anumite domenii devine sesizabilă și chiar necesară, apărând, astfel, o multitudine de "unelte" tehnologice cu ajutorul cărora ne putem securiza datele.

Une dintre metodele cu care putem ascunde date poate fi și **steganografia**, termen care provine din limba greacă, unde **steganos** înseamnă *ascuns* și **graph** înseamnă *scris*. Rezumând, am putea spune că steganografia este știința sau arta de a scrie mesaje ascunse astfel încât existența lor să fie cunoscută numai de către destinatar și expeditor.

În lumea digitală, există mai multe tipuri de steganografie, însă, ne vom rezuma doar la cea care presupune **alterarea imaginilor** pentru a ascunde date "**în plină vedere**". Dintre metodele steganografice existente, a fost aleasă cea a bitului cel mai puțin semnificativ (**LSB Steganography**) datorită ușurinței de implementare a acesteia într-un program.

Proiectul presupune, în primul rând, testarea legitimității metodei de a **ascunde** mesaje sau orice alt tip de date serializabil, în pixelii unei **imagini**. S-a dorit, de asemenea, realizarea operației cu un minim de **impact vizual**, astfel încât încifrarea unui mesaj într-o imagine să nu denatureze calitatea acesteia într-un mod vizibil de către ochiul uman.

În urma implementării unei aplicații simpliste care automatizează procesul menționat anterior, s-a observat că se pot ascunde mesaje de dimensiuni considerabile în imagini pe care majoritatea le considerăm normale. De asemenea, timpul necesar pentru a încifra un mesaj într-o imagine este absolut neglijabil, fiind, în cele mai complexe cazuri (vorbind despre sute de mii sau milioane de caractere) de ordinul secundelor.

Cuvinte cheie: **imagini, mesaj, ascundere**

1 Introducere

Observând numărul considerabil de domenii în care poate fi folosită steganografia atât pentru transmitere nesensizată de date, fie ea bine sau rău intenționată, cât și pentru așa zisa imprimare a copyright-ului în variate produse digitale, am decis că o aprofundare a acestui subiect ar fi benefică din mai multe puncte de vedere.

Ne propunem să creăm o aplicație care poate codifica un mesaj într-o imagine fără a face modificări asupra acesteia, care ar putea fi sesizate de către ochiul uman. În acest caz, termenul "mesaj" este unul relativ, încât acesta poate fi un **string**, poate fi o **image**, sau chiar **orice** fel de **fișier**.

Steganografia nu trebuie confundată cu **criptografia**. Aceasta din urmă face ca un mesaj să devină indescifrabil, dar existența lui este vizibilă, iar steganografia "ascunde" existența mesajului și nu mesajul în sine.

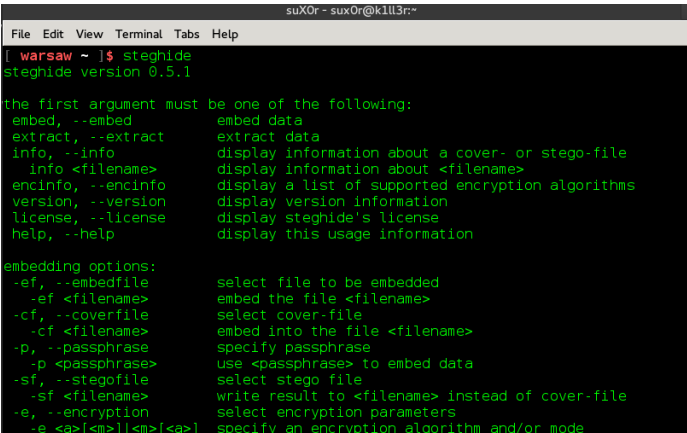
Câteva zone în care întâlnim steganografia:

- **Domeniul militar:** zonă în care transmiterea de informații trebuie să fie făcută într-un cerc extrem de select și într-un mod securizat, întrucât interceptarea și/sau descifrarea informațiilor poate pune în pericol o întreagă națiune.
- **Transmiterea virusilor** prin intermediul imaginilor sau videoclipurilor, care, odată ajunse pe un dispozitiv încep să deterioreze informațiile stocate sau să le transmită mai departe.
- **Media:** amintim de noțiunea de watermark, ce presupune prezența unui mesaj ascuns sau semi-ascuns privitorului, cu scopul de a conferi o protecție asupra drepturilor de autor, dar și o cale de a reda originalitate unor proiecte.
- Steganografia poate fi folosită și într-un mod mai "casual", în sensul ascunderii **documentelor** de o importanță mai mare în plină vedere, în imagini obișnuite, atât timp cât noi deținem metoda decodificării mesajului.

2 Metode existente

În momentul de față există în lumea informatică mai multe metode de aplicare a stenografiei, cea mai simplă fiind menționată anterior, celelalte fiind: metode statistice, metode în frecvență, distorționarea imaginii, etc.

Printre aplicațiile deja existente și disponibile pentru toată lumea pentru a ascunde mesaje prin intermediul steganografiei imaginilor, regăsim: **Steghide**, **Xiao Steganography**, **OpenStego**, majoritatea folosind chiar și concepte criptografice pentru a securiza mesajul ascuns în cazul în care este detectat de către o persoană terță.



```
suX0r - suX0r@k1ll3r:~
File Edit View Terminal Tabs Help
[ warsaw ~ ]$ steghide
steghide version 0.5.1

the first argument must be one of the following:
embed, --embed          embed data
extract, --extract      extract data
info, --info            display information about a cover- or stego-file
info <filename>        display information about <filename>
encinfo, --encinfo      display a list of supported encryption algorithms
version, --version      display version information
license, --license      display steghide's license
help, --help            display this usage information

embedding options:
-ef, --embedfile        select file to be embedded
-ef <filename>          embed the file <filename>
-cf, --coverfile        select cover-file
-cf <filename>          embed into the file <filename>
-p, --passphrase        specify passphrase
-p <passphrase>         use <passphrase> to embed data
-sf, --stegofile        select stego file
-sf <filename>          write result to <filename> instead of cover-file
-e, --encryption        select encryption parameters
-e <a>[<m>][<m>][<a>]   specify an encryption algorithm and/or mode
```

Figure 1: Steghide - linie de comandă

3 Descrierea tehnică a soluției

3.1 Soluția propusă

Pentru acest proiect, se va folosi Steganografia **LSB** - Least Significant Bit, o metodă relativ simplă care presupune alterarea celui mai puțin semnificativ bit (dacă acesta nu corespunde deja cu bitul mesajului pe care vrem să-l ascundem) al fiecărui canal de culoare din pixelii imaginii.

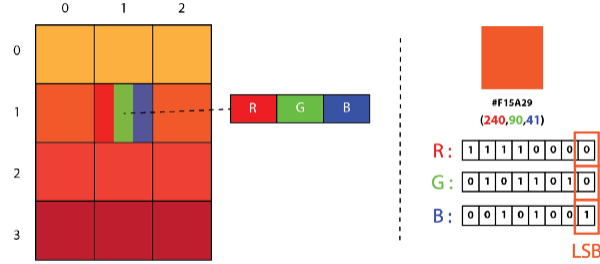


Figure 2: Reprezentare grafică a metodei LSB

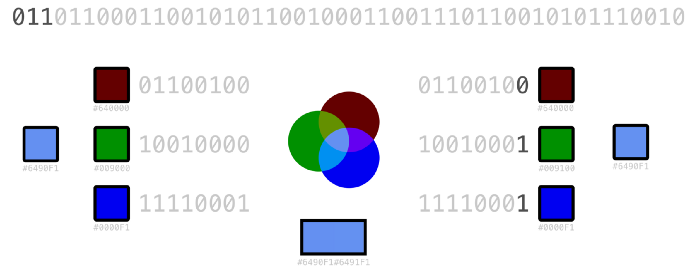


Figure 3: Corelarea modificărilor cu un mesaj

Modificând cel mai nesemnificativ bit al fiecărui canal, este evident că se produc anumite schimbări ale culorilor la nivel de pixel, însă, acestea sunt total **inesizabile** de către ochiul uman, diferențele de intensitate fiind mult prea mici.

Astfel, operațiunea de ascundere a unui mesaj într-o imagine se reduce la simpla iterare prin pixelii acesteia, modificând cel mai nesemnificativ bit al fiecărui canal de culoare acolo unde este cazul.

Algorithm 1: Algoritm ascundere mesaj

```
iterator ← 0;
for i ← 0 to inaltime_imagine do
  for j ← 0 to latime_imagine do
    if iterator < lungime_mesaĳ then
      bit_8_canal_R = mesaj_serializat[iterator];
      iterator ← iterator + 1;
      repetăm pentru fiecare canal;
    if iterator == lungime_mesaĳ then
      stop;
```

Deducem imediat că mesajul poate fi **decodat** iterând prin pixelii imaginii și extrăgând fiecare ultim bit ale canalelor de culoare. Ulterior, șirul de biți rezultat poate fi reconvertit într-un string.

Algorithm 2: Algoritm decodare mesaj

```
iterator  $\leftarrow$  0;
for  $i \leftarrow 0$  to inaltime_imagine do
    for  $j \leftarrow 0$  to latime_imagine do
        mesaj_decodat += bit_8_canal_R;
        if ultimul byte din mesajul decodat este echivalent cu caracterul de control then
            stop;
        else
            repetăm pentru fiecare canal
    for bit în mesaj_decodat do
        string_temporar += bit;
        if lungime_string_temporar % 8 == 0 then
            string_decodat += string_temporar;
            resetăm string_temporar
return string_decodat
```

3.2 Detalii legate de implementare

Implementarea algoritmilor menționați anterior a fost făcută în limbajul **Python**, iar, pentru lucrul cu imagini s-a folosit modulul **Pillow**.

Putem separa procesul de implementare în următoarele etape:

1. Citirea imaginii și mesajului care urmează să fie codificat în aceasta (payload).
Comandă pentru citirea imaginii: ***image = Image.open("numeImagine.extensie")***.
2. Serializarea mesajului - transformarea acestuia într-un șir de biți.

```
byte = bin(ord(caracter_de_convertit))[2:]
byte = '%08d'%int(byte)
mesaj_serializat += byte
```

Cu mențiunea că funcția **ord** returnează un întreg ce reprezintă echivalentul caracterului în Unicode, **bin** returnează un string în care regăsim configurația în baza 2 a caracterului, iar **+=** nu face decât să concateneze ce se află după operator într-un șir.

Obținem, astfel, o codificare pe biți a mesajului scrisă în string-ul '**mesaj_serializat**'.

3. Se verifică dacă mesajul poate fi ascuns în imagine.
Presupunând că imaginea este de dimensiune $W \times H$ și ținând cont că noi putem altera 3 biți la fiecare pixel (3 canale - R,G, B per pixel), numărul maxim de biți pe care putem codifica mesajul este: $3 \times W \times H$
4. Se alterează LSB conform metodei descrise anterior în **Algorithm 1**. Operația se repetă de atâtea ori câți biți avem în **payload**.
5. Se salvează imaginea cu mesajul încifrat în aceasta.

Evident, pentru partea de **decodare** se va face doar o parcurgere a canalelor de culoare ale fiecărui pixel și se va adăuga fiecare LSB într-un șir de biți ce urmează să fie reconvertit către tipul inițial al mesajului. Metoda este descrisă în **Algorithm 2**.

Partea de **GUI** a aplicației presupune oferirea posibilității de a selecta imaginea și mesajul care să fie ascuns în aceasta de către utilizator, decodarea unui mesaj dintr-o imagine, precum și o pagină în care se pot vizualiza **histogramele** unei **imagini normale** și ale unei **imagini** care conține mesajul **codificat** prin metodele descrise anterior.

Datorită flexibilității limbajului Python am extins posibilitatea ascunderii informațiilor pe mai mult de un bit (se va putea alege între 1,2,4 sau 8 biți) la fiecare canal. Automat, această practică nu respectă principiul steganografiei LSB.

Motivația acestei aditii a fost posibilitatea de a folosi mai mulți biți în procesul de încifrare pentru a vedea experimental cât de mult poate fi alterată o imagine din punct de vedere vizual, s-a dorit găsirea unui "prag" care să ne spună cât de multă informație putem ascunde fără a da semne de îndoială.

4 Rezultate experimentale

Pentru a testa eficiența mini aplicației proiectate s-au urmărit următoarele aspecte:

- cât de mult se modifică imaginea în urma încifrării mesajului (vizual).
- cât de repede se execută algoritmul.

Alterarea vizuala a unei imagini a fost destul de greu de urmărit cu ochiul liber, chiar și la mesaje de lungimi uriașe (sute de mii de caractere) când venea vorba de imagini cu o rezoluție modestă (**ex:** 1440×810).

Astfel, s-a ales să se implementeze o metodă prin care să putem vedea **histograma** imaginii **originale** și a imaginii **alterate**, precum și suprapunerea celor două pentru a observa mai ușor intervalele în care există **diferențe**. **Pillow** ne-a oferit posibilitatea de a extrage direct histograma din imaginea dată ca sursă, urmând să o separăm în 3, pentru fiecare canal, iar pentru plotare s-a folosit **matplotlib**.

În ceea ce privește timpul de execuție, s-a observat că în cel mai defavorabil caz se poate vorbi de câteva secunde (atunci când dorim să ascundem mesaje de dimensiuni uriașe).

Ca referință, imaginea pe care au avut loc încifrările are dimensiunea 1440×810 , iar măsurătorile au fost făcute cu modulul **datetime**.

Număr biți/canal utilizați	Dimensiune mesaj(octeți)	Timp
1	50	0:00:00.008997
1	15000	0:00:00.193999
1	430000	0:00:01.232998
2	430000	0:00:00.622998
4	430000	0:00:00.317995
8	430000	0:00:00.157998

Se observă faptul că timpul de execuție este oarecum invers proporțional cu numărul de biți alterați pe fiecare canal. Un timp foarte bun este obținut atunci când folosim toți biții canalului pentru a codifica mesajul, însă, imaginea este complet modificată în ceea ce privește vizualul.

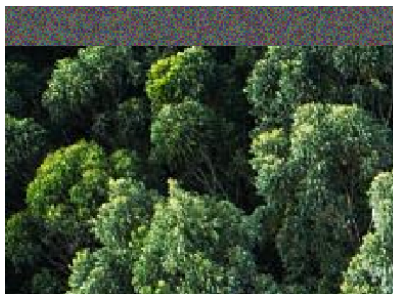


Figure 4: Imaginea cu mesajul încifrat pe toți biții dintr-un canal

Se poate observa, fără urmă de îndoială, că în practică nu ar avea sens să se folosească o astfel de codificare steganografică pe toți biții unui canal de culoare.

Diferența dintre cele două imagini este observată și în histogramele generate din aplicație:

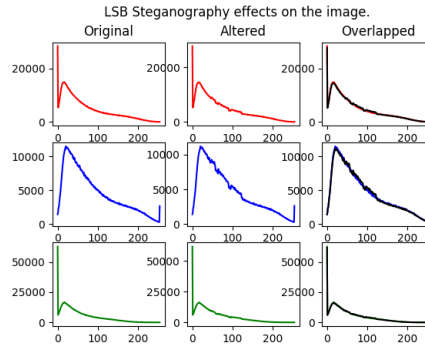


Figure 5: Histogramele aferente scenariului descris anterior

În cazul în care folosim metoda clasică, și anume folosirea unui singur bit, cel mai nesemnificativ dintr-un singur canal de culoare pentru a codifica un mesaj de 430000 de caractere, imaginea alterată arată exact la fel pentru ochiul uman, iar histogramele se suprapun cu diferențe mai mici decât în scenariul anterior.

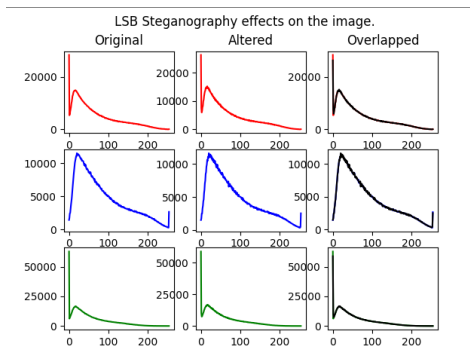


Figure 6: Histograme rezultate în urma încifrării cu 1 bit/canal

S-a observat că atunci când folosim 1-2 biți/canal, diferențele dintre imaginea originală și cea alterată sunt prea mici pentru a putea fi sesizate de către ochiul uman, atât timp cât imaginea are o rezoluție rezonabilă. Când se folosesc 4 biți diferențele se pot sesiza cu foarte puțină atenție (**Figura 7**), iar la 8 biți rezultatul a fost prezentat în **Figura 4**.

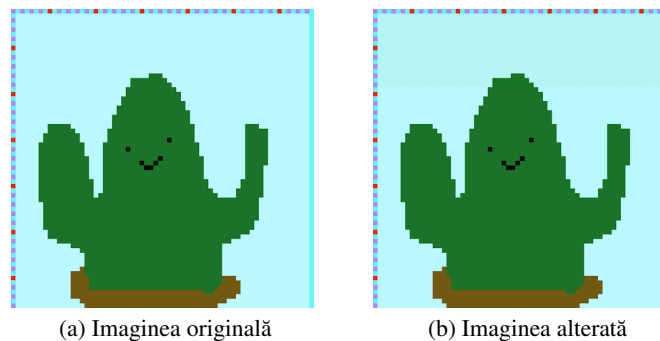


Figure 7: Encoding cu 4 biți/canal de culoare

5 Concluzii

În urma testării algoritmului pe un set mai larg de imagini s-a ajuns la consensul că ascunderea unui mesaj pe mai mult de 2 biți per canal de culoare este detectabilă de către ochiul uman și nu ar trebui să fie folosită în practică. Faptul că timpul de prelucrare a imaginii este mai mic cu cât creștem numărul de biți este total irelevant, având în vedere că oricum este vorba despre timpi de ordinul secundelor, în cele mai nefavorabile cazuri.

Aplicația noastră permite ascunderea mesajelor într-un mod total nedetectabil de către ochiul uman (dacă ne rezumăm la codare pe 1-2 biți/canal), însă, probabil că un sistem informatic avansat sau un specialist în prelucrare de imagini ar detecta faptul că este ceva în neregulă cu imaginea.

Aplicația ar putea fi îmbunătățită prin elaborarea unui algoritm care nu înlocuiește biții fiecărui canal de culoare într-un mod continuu, scăzând astfel considerabil șansa de a fi detectată alterarea. De asemenea, se putea introduce și o criptare a mesajului ascuns, pentru dublarea protecției, însă, acest lucru nu a fost realizat pentru că nu aducea un plus de valoare în ceea ce privește prelucrarea de imagini.

De menționat ar fi și faptul că aplicația poate fi extinsă pentru a ascunde orice tip de fișier serializabil, atât timp cât se implementează și metodele de reconstruire a fișierelor din forma binară.

Referințe

- [1] Harpreet Kaur and Jyoti Rani (2016) A Survey on different techniques of steganography. CSE Department, GZSCCET Bathinda, Punjab, India
- [2] <https://en.wikipedia.org/wiki/Steganography>
- [3] <https://www.ukessays.com/essays/computer-science/steganography-uses-methods-tools-3250.php>
- [4] <http://www.eiron.net/thesis>
- [5] <https://itnext.io/steganography-101-lsb-introduction-with-python-4c4803e08041>
- [6] <https://pillow.readthedocs.io/en/stable/>
- [7] <https://matplotlib.org/contents.html>