Cuprins

Introducere

Comunicarea electronică este din ce în ce mai susceptibilă la a fi interceptată sau manipulată cu scop malițios. Problemele de securitate și confidențialitate au fost abordate de cele mai multe ori prin instrumentele de criptografie. Mesajele sunt criptate astfel încât numai persoanele cărora le-a fost destinat mesajul să îi poată verifica integritatea, autenticitatea și ulterior, să îl citească. Criptografia este un domeniu matur, a cărui fundamente sunt matematica și decenii de dezvoltare.

Mesajele criptate sunt însă evidente, iar atunci când sunt interceptate, este evident faptul că emițătorul și receptorul comunică în secret. Din acest motiv, steganografia a apărut ca alternativă pentru criptografie. Criptarea mesajelor este înlocuită cu ascunderea lor în obiecte obișnuite, astfel încât prezența mesajelor să nu fie vizibilă. Putem considera steganografia fezabilă atunci când nu se dorește atragerea atenției nedorite.

Steganografia are originile în antichitate. Autorul cuvântului este Trithemius, cel care a scris de asemenea și publicațiile Polygraphia și Steganographia. Termenul derivă din cuvintele grecești *steganos* (a ascunde) și *graphia* (scris). Așadar, ”steganografia este arta comunicării ascunse. Însăși existența mesajului este secretă.”[1]

Un exemplu de steganografie aplicată deseori întâlnit este exemplul lui Herodotus, care menționează un sclav trimis de către stăpânul său către orașul Ionian Miletus. Acesta a tatuat pe capul sclavului un mesaj, urmând ca acestuia să îi crească părul, ascunzând astfel existența mesajului. Odată ajuns in orasul Miletus, sclavul și-a ras părul pentru a-i arăta mesajul lui Aristagoras, determinându-l pe acesta să pornească o revoltă împotriva regelui persan.

În anii recenți, steganografia a fost folosită de gruparea teroristă Al Qaeda, când asupra unui tânar de 22 de ani au fost găsite mai multe carduri de memorie, ce conțineau videoclipuri cu conținut explicit. ”Săptămâni mai târziu, după eforturi grele de a sparge o parolă și un software pentru a face fișierele aproape invizibile, investigatorii germani au găsit codificate în videoclipuri o comoară; mai mult de 100 de documente Al

Qaeda ce includeau traseul pe care unele din cele mai insolente planuri urmau să îl ia, cât și o hartă a operațiunilor viitoare.”[2]

Lucrarea are ca obiectiv prezentarea steganografiei ca domeniu adiacent criptografiei, conceptele presupuse de aceasta și metodele pe care le oferă. În cadrul lucrării sunt ilustrați 4 algoritmi ca soluții la comunicarea confidențială, în funcție de contextul acesteia.

În continuare, capitolul ”Contribuții” prezintă în câteva cuvinte principalele realizări personale. În capitolul ”Metrici” sunt descrise 2 metrici uzuale utilizate în domeniu, pentru măsurarea calității imaginii. Capitolul ”Descrierea soluției” este împărțit în 4 subcapitole, fiecare corespunzând unuia din cei 4 algoritmi, care la rândul lor sunt prezentați sub forma unei scurte descrieri despre aceștia, urmat de modul de funcționare (încorporarea și extragerea mesajului), iar la final, o scurtă analiză a performanțelor. Capitolul ”Concluzii” reprezintă un rezumat al lucrării, ce include și o scurtă opinie personală.

Contribuții

Contribuțiile personale sunt următoarele:

- Alcătuirea unui rezumat al domeniului
- Alegerea algoritmilor și implementarea acestora ca *proof of concept*
- Adaptarea algoritmilor la tehnologia utilizată
- Analiza metodelor prezentate
- Comparația acestora

Metrici

Pentru măsurarea calității imaginii originale și a imaginii rezultate, am utilizat 2 metrici, și anume, Mean Square Error (MSE) și Peak Signal to Noise Ratio (PSNR).

MSE reprezintă rata de eroare totală, raportată la imaginea originală. Cu cât este mai aproape de 0, cu atât este mai bună, deci stego-imaginea și imaginea sunt mai asemănătoare. Ea se calculează astfel:

$$MSE = \frac{1}{M * N}\sum_{x=1}^{N}\sum_{y=1}^{M}[I(x,y) - I'(x,y)]^2$$

unde:

$$N,\ M\ - dimensiunile\ imaginii$$

$$I,\ I' - imaginile\ comparate$$

$$I(x,\ y) - valoarea\ pixelului\ cu\ coordonatele\ x,\ y$$

PSNR este metrica opusă MSE, măsurând gradul de similaritate dintre imagini. Ea exprimă raportul dintre valoarea maximă a semnalului (a pixelilor imaginii, în cazul nostru), și puterea zgomotului ce afectează calitatea lui. Cu cât este mai mare, cu atât imaginile sunt mai similare. Ea se calculează astfel:

$$PSNR = 10 * \log_{10}\left(\frac{255^2}{MSE}\right)$$

Descrierea soluțiilor

Problema este abordată prin algoritmii prezentați în continuare. Implementarea acestora s-a realizat cu ajutorul bibliotecii OpenCV[3], o bibliotecă comună și stabilă ce facilitează lucrul cu imagini, oferind diverse funcții prin care se poate interacționa cu acestea la nivel de pixeli.

Algoritmi sunt aplicați pe imagini grayscale, dar pot fi extinși cu ușurință și pentru imagini color.

Pentru testare și analiză, s-au utilizat următoarele imagini:

”dog_256x256.png” – Imagine cu dimensiunile de 256 * 256,

”dog.png” – Imagine cu dimensiunile de 768 * 512, grayscale

1. LSB (**L**east **S**ignificant **B**it Replacement):

În steganografia imaginilor, majoritatea algoritmilor urmăresc modificarea informației care are cel mai mic impact asupra imaginii, cu scopul de păstra stego-imaginea cât mai aproape de imaginea originală. Un algoritm propus în acest sens este **LSB**, care, de altfel, este și algoritmul de bază. Principiul utilizat în această schemă va fi folosit în continuare de către câțiva din algoritmii prezentați.

1.1 Algoritmul:

Schema propusă constă în modificarea celui mai nesemnificativ bit (LSB) al fiecărui pixel al imaginii purtător. ”Ascunderea datelor în LSB ai pixelilor unei imagini grayscale este o metodă comună ce utilizează caracteristica ochiului uman de insensibilitate la modificări mici în imagini.”[4]

Trebuie menționat faptul că cu cât mai mulți LSB ai unui pixel sunt modificați, cu atât stego-imaginea rezultată va fi mai distorsionată.

Algoritmul are 2 etape: încorporarea și extragerea mesajului.

1.1.1 Încorporarea mesajului:

Încorporarea mesajului este simplă. Imaginea purtătoare este citită, odată cu mesajul ce trebuie ascuns. Mesajul este transformat într-un șir de biți, la începutul căruia se atașează lungimea mesajului, tot sub formă de șir de biți, deoarece este necesar ca receptorul mesajului să poată afla lungimea mesajului. Din acest motiv, presupunem că lungimea șirului se exprimă într-un număr de biți pe care îl cunosc atât emițătorul, cât și receptorul. Pasul următor este traversarea imaginii, pixel cu pixel, timp în care fiecărui pixel îi sunt modificați numărul de LSB desemnat de algoritm.

Imaginea rezultată este stego-imaginea, care este salvată într-un format lossless (.png, spre exemplu). Figura 1 ilustrează etapele încorporării.

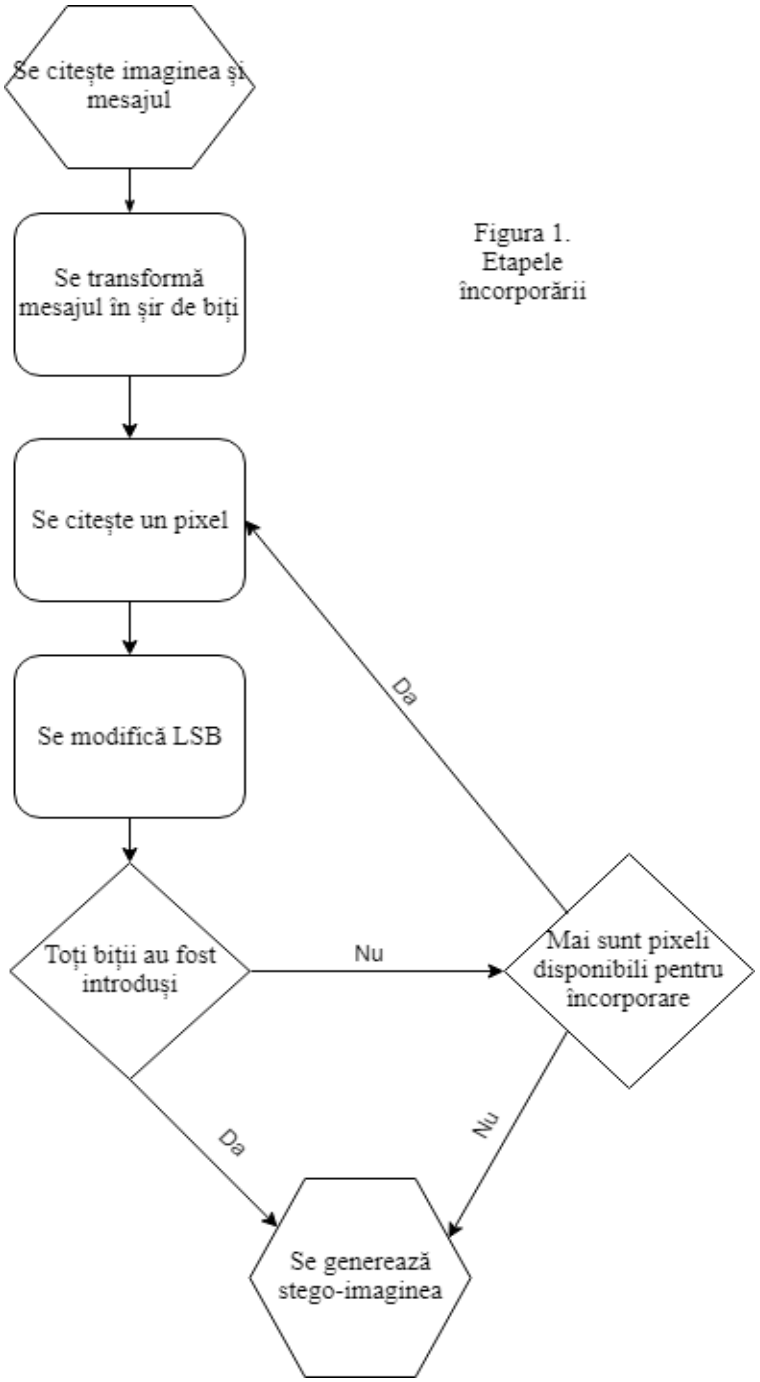

Figura 1. Etapele încorporării

Algoritmul în pseudocod:

Input: imaginea I, mesajul M

Output: stego-imaginea S

```
messageLength = |M|;
S = I;
secretBits = toBits(messageLength) +  toBits(M);
i = 0, j = 0;
while(i < S.rows && secretBits.size() > 0){
       while(j < S.cols && secretBits.size() > 0){
              pixel = S.at(i, j)
              pixelBits = toBits(pixel);
              pixelBits[0] = secretBits.top();
              secretBits.pop();
              S.at(i, j) = toPixel(pixelBits);
              j++;
       }
       j = 0;
       i++;
}
return S;
```

1.1.2 Extragerea mesajului:

Extragerea mesajului este intuitivă în acest caz, cu excepția faptului că trebuie mai întâi extrasă lungimea mesajului. Atât lungimea, cât și mesajul însuși, se extrag prin același procedeu: se ia fiecare pixel în parte, iar LSB-ul este pus într-un șir de biți. Dupa ce se obține lungimea, se iau pixelii neprocesați, de la primul pixel neprocesat, și se extrag LSB până când lungimea mesajului a fost atinsă sau au fost procesați toți pixelii. După extragere, se afișează mesajul. Figura 2 ilustrează procesul de extragere.

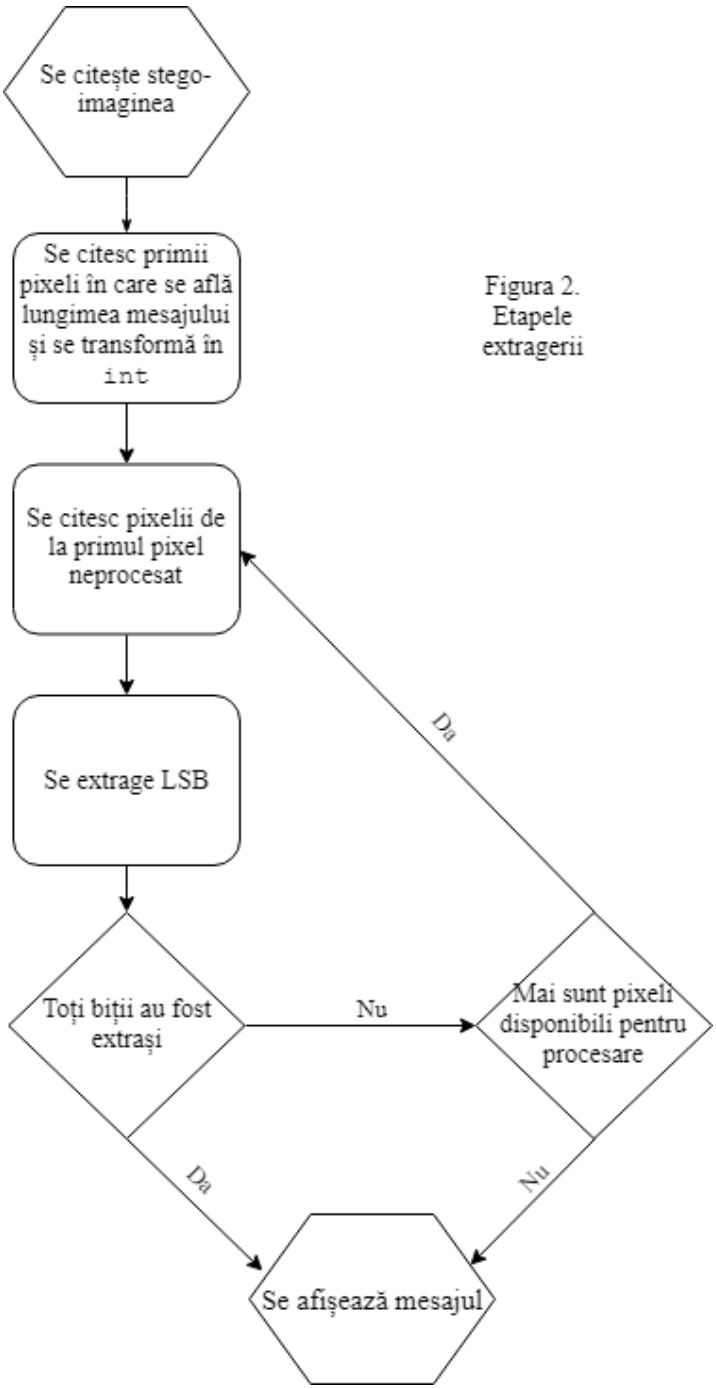

Figura 2. Etapele extragerii

Algoritmul în pseudocod:

Input: stego-imaginea S

Output: mesajul ascuns M

```
i = 0, j = 0, index = 0;
int[32] messageLengthBits;
while(i < S.rows && index < 32){
       while(j < S.cols && index < 32){
              pixel = S.at(i, j);
              pixelBits = toBits(pixel);
              messageLengthBits[index++] =
                                    pixelBits[0];
              j++;
       }
       lastUnprocessedPixel= j;
       j = 0;
       i++;
}
j = lastUnprocessedPixel;
messageLength = toInt(messageLengthBits);
index = 0;
int[messageLength] messageBits;
while(i < S.rows && messageLength * 8 < 32){
       while(j < S.cols && messageLength * 8 < 32){
              pixel = S.at(i, j);
              pixelBits = toBits(pixel);
              messageBits [index++] = pixelBits[0];
              j++;
              }
       j = 0;
       i++;
       }
return toMessage(messageBits);
```

1.3 Rezultate și observații:

Tabel 1 – Capacitatea de încorporare imaginii utilizate

Imagine purtător	Dimensiune	1 LSB	2 LSB	4 LSB
dog.png	768 * 512	768 * 512	768 * 512 * 2	768 * 512 * 4

Pentru analiza acestui algoritm, am aplicat 3 iterații ale acestui algoritm, modificând primul LSB, primii 2 LSB, respectiv primii 4 LSB. Figurile 3 – 3.3 ilustrează rezultatele obținute în cele 3 cazuri (folosind ” dog_256x256.png”). Textul ascuns în acest context este un capitol dintr-o carte, având 16,205 caractere, adică 129,640 biți.

Figura 3 Imaginea originală

Figura 3.1 Stego-imaginea 1 LSB

Figura 3.2 Stego-imaginea 2 LSB

Figura 3.3 Stego-imaginea 4 LSB

După cum se observă, comparativ cu imaginea originală, stego-imaginea 1 LSB, respectiv 2 LSB sunt foarte asemănătoare. În cazul 4 LSB, se observă deja, după cum a fost menționat, că zonele uniforme ale imaginii originale sunt alterate vizibil în stego-imagine. Datorită capacității de încorporare reduse ale imaginii purtător, în cazul 1 LSB mesajul nu a putut fi introdus în întregime. 2 LSB și 4 LSB au avut suficient spațiu pentru a stoca întregul mesaj. De aceea, în funcție de dimensiunea mesajului, imaginea purtător trebuie aleasă corespunzător.

Tabel 2 – Metricile obținute pentru iterațiile 1 LSB, 2 LSB, 4 LSB

Iterație	MSE	PSNR
1 LSB	0.00000763498246669769	99.3027191162109375
2 LSB	0.00003479700535535812	92.71538543701171875
4 LSB	0.00032401899807155132	83.0251007080078125

Figura 4. – De la stânga la dreapta, imaginea originală, 1 LSB, 2 LSB, 4

Conform tabelului 2, rata de eroare pentru 1 LSB și 2 LSB este mică, deci raportul semnal-zgomot este mare. Aceste două măsuri ne indică faptul că diferențele dintre stego-imagini și imaginea purtătoare sunt foarte mici. Pe de altă parte, pentru 4 LSB, rata de eroare crește considerabil, iar raportul semnal-zgomot scade în aceeași măsură. După cum se observă în figura 4, zona uniformă prezintă diferențe observabile față de imaginea originală.

Algoritmul nu rezistă atacurilor. O simplă compresie, un canal de comunicare instabil sau manipularea stego-imaginii în timpul transmisiei poate determina pierderea mesajului.

Putem concluziona astfel că algoritmul este unul rudimentar. El doar implementează un principiu de bază. Atunci când este folosit, imaginea purtătoare trebuie aleasă în funcție de dimensiunile mesajului și de numărul de LSB pe care dorim să îl schimbăm.

2 PVD (**P**ixel **V**alue **D**ifferencing):

Acest algoritm propune o metodă eficientă de ascundere a mesajului prin adaptarea numărului de LSB pe care îi modificăm, în funcție de gradul de uniformitate al zonei pe care o procesăm. Schema a fost dezvoltată cu scopul de a produce rezultate mai imperceptibile decât rezultatele algoritmului LSB.

2.1 Algoritm:

Algoritmul propune divizarea imaginii în blocuri de 2 pixeli consecutivi, nesuprapuse. Fiecare bloc este clasificat în funcție de diferența valorilor pixelilor. ”O diferență mică indică faptul că blocul este într-o zonă uniformă, iar o diferență mare indică faptul că blocul este într-o zonă neuniformă.”[4]

”Nu toți pixelii unei imagini pot tolera același număr de biți schimbați, fără să atragă atenția observatorului. [...] Schimbări în nuanțele de gri ale pixelilor aflați în zonele uniforme din imagini sunt mai ușor de distins de ochii umani.”[4]

Așadar, în metoda propusă încorporăm mai mulți biți în zone neuniforme, pentru a obține o stego-imagine mai greu de distins de original.

2.1.1 Încorporarea:

Asemenea algoritmului LSB, mesajul este transformat în șir biți.

Procesul începe prin calcularea diferenței d a fiecărui bloc de pixeli consecutivi, p_i și p_{i+1}. Presupunând că valorile pixelilor sunt g_i și g_{i+1}, atunci d se calculează prin diferența $g_i - g_{i+1}$. Un bloc al cărei diferență este aproape de 0 este considerat bloc uniform, iar dacă diferența este aproape de -255 sau 255, atunci este considerat bloc neuniform. Prin simetrie, considerăm doar valorile absolute ale d. Blocurile sunt clasificate după o mulțime de intervale R contigue. Numim această clasificare *cuantizare*.

$$R = \{R_i \mid i = 1, 2, \dots n\}$$

$$R_i = [l_i, u_i]; l_0 = 0, u_n = 255, |R_i| = u_i - l_i + 1$$

În acest algoritm, am ales ca lățimea intervalelor R_i să fie o putere a lui 2. ”Această restricție facilitează încorporarea datelor binare. Intervalele selectate sunt bazate pe capacitatea vizuală umană menționată anterior. Lățimile intervalelor ce reprezintă diferențe ale blocurilor uniforme sunt mai mici, pe când cele ce reprezintă diferențe ale blocurilor neuniforme sunt mai mari.”[4]

Toate diferențele care aparțin unui interval sunt considerate *apropiate*. Asta înseamnă că dacă o diferență este înlocuită cu o diferență din același interval, schimbarea nu este perceptibilă de ochii umani. În acest algoritm, modificăm valorile pixelilor schimbând diferența menționată cu o diferență din același interval.

După calculul diferenței și cuantizarea blocului, numărul de biți *n* ce pot fi încorporați este calculat după formula $n = log_2(u_k - l_k + 1)$ (*k* fiind indexul intervalului de care aparține diferența). Deoarece lățimea intervalului este o putere a lui 2, numărul *n* este număr întreg. Se alcătuie un șir de *n* biți consecutivi din șirul *S* de biți ai mesajului. O nouă diferență *d'* se calculează, după următoarea formulă:

$$d' = \begin{cases} l_k + b, & d \geq 0 \\ -(l_k + b), & d < 0' \end{cases}$$

$$b = valoarea\ subșirului\ din\ S$$

d' aparține intervalului $[l_k, u_k]$ deoarece *b* se află în intervalul $[0, u_k - l_k]$. Șirul *b* este încorporat prin calculul invers din care se obțin noi valori din *d'*. Calculul invers este definit de funcția $f((g_i, g_{i+1}), m)$.

$$f\big((g_i, g_{i+1}), m\big) = (g_i', g_{i+1}')$$

$$= \begin{cases} g_i - ceil\left(\frac{m}{2}\right), g_{i+1} + floor\left(\frac{m}{2}\right), pentru\ d\ impar \\ g_i - floor\left(\frac{m}{2}\right), g_{i+1} + ceil\left(\frac{m}{2}\right), pentru\ d\ par \end{cases}$$

$$m = d - d'$$

Calculul de mai sus poate produce însă valori care ies din intervalul [0, 255] dacă *d'* are valori prea mici, sau prea mari. De aceea, algoritmul are o etapă de verificare, în care produce o pereche $(\hat{g}_i, \hat{g}_{i+1})$ prin calculul invers $f((g_i, g_{i+1}), u_k - d)$. Dacă valorile $\hat{g}_i$ și $\hat{g}_{i+1}$ ies din intervalul [0, 255], atunci considerăm blocul nepotrivit pentru încorporare. Figura 3 ilustrează procesul de încorporare.

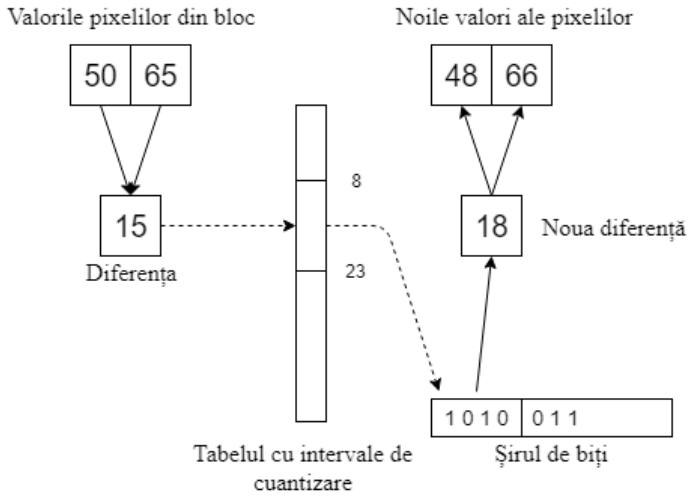

Figura 3. Procesul de încorporare

Figura 4 ilustrează pașii algoritmului.

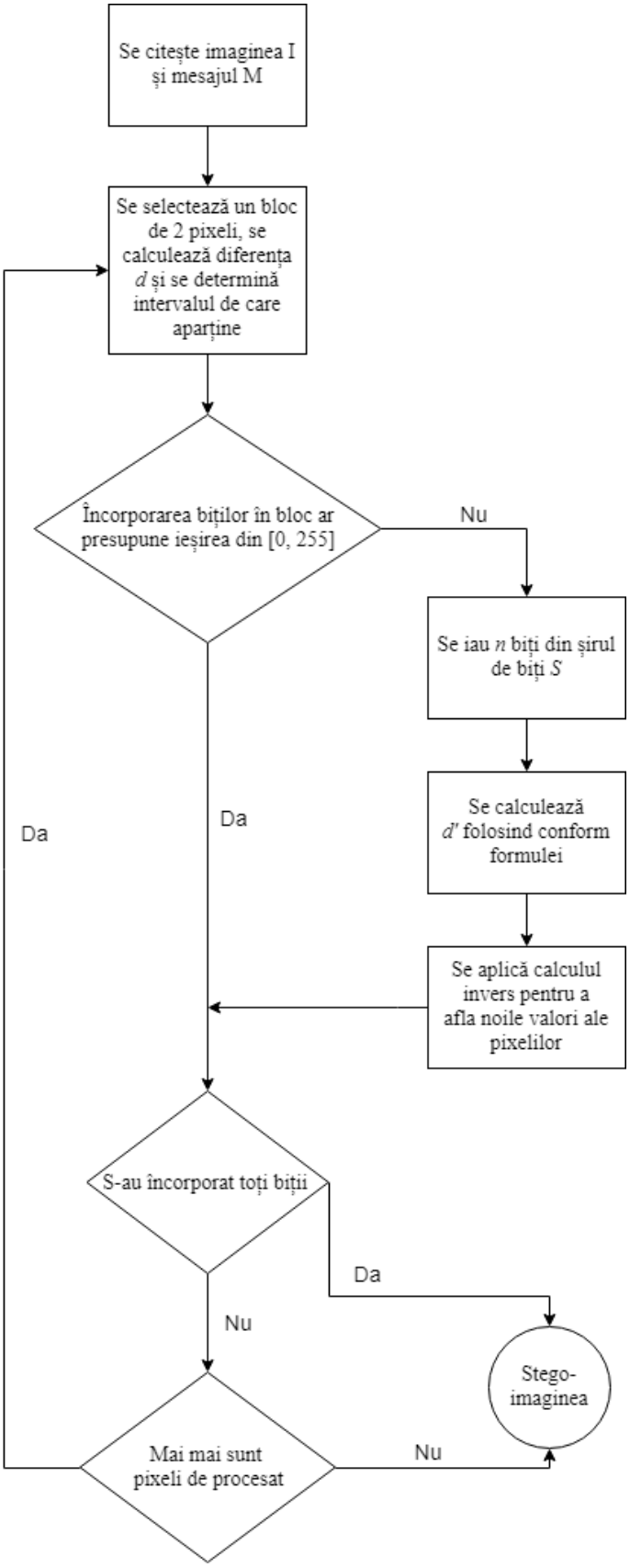

Figura 4. Încorporarea mesajului

Algoritmul în pseudocod:

Input: imaginea purtător I, mesajul M

Output: stego-imaginea S

```
messageLength = |M|;
S = I;
secretBits = toBits(messageLength) +  toBits(M);
i = 0, j = 0;
while(i < S.rows && secretBits.size() > 0){
       while(j < S.cols && secretBits.size() > 0){
              p1 = S.at(i, j);
              p2 = S.at(i, j+1);
              d = p1 – p2;
              interval = getInterval(d);
              check1, check2 = computePixelValues(p1, p2, d –
                                                  interval.u);
              if(check1 < 0 || check1 > 255
              || check2 < 0 || check2 > 255){
                     j+=2;
                     continue;
              }
              n = getNumOfBits(interval);
              substream = getSubStream(secretBits, n);
              value = toValue(substream);
              d’ = getNewDifference(d, value, interval);
              m = d – d’;
              pixel1, pixel2 = computePixelValues(p1, p2, m);
              S.at(i, j) = pixel1;
              S.at(i, j+1) = pixel2;
              j+=2;
              }
       j = 0;
       i++;
}
return S;
```

2.1.2 Extragerea mesajului:

Asemenea algoritmului LSB, extragerea mesajului se face în două etape: obținerea lungimii mesajului și extragerea mesajului propriu-zis. Ambele se fac prin același proces. Imaginea este împarțită în blocuri de câte 2 pixeli nesuprapuse, apoi se calculează diferența d^*. Deoarece există posibilitatea ca blocul să nu fi fost folosit, se aplică aceeași verificare $f((g_i^*, g_{i+1}^*), u_k - d^*)$ pentru a obține $(\hat{g}_i^*, \hat{g}_{i+1}^*)$. Dacă cele două valori nu ies din intervalul [0, 255], atunci blocul a fost folosit, și putem extrage valoarea b introdusă la pasul anterior. Figura 5 ilustrează pașii extragerii.

$$b = \begin{cases} d^* - l_k, & d^* \geq 0 \\ -d^* - l_k, & d^* \leq 0 \end{cases}$$

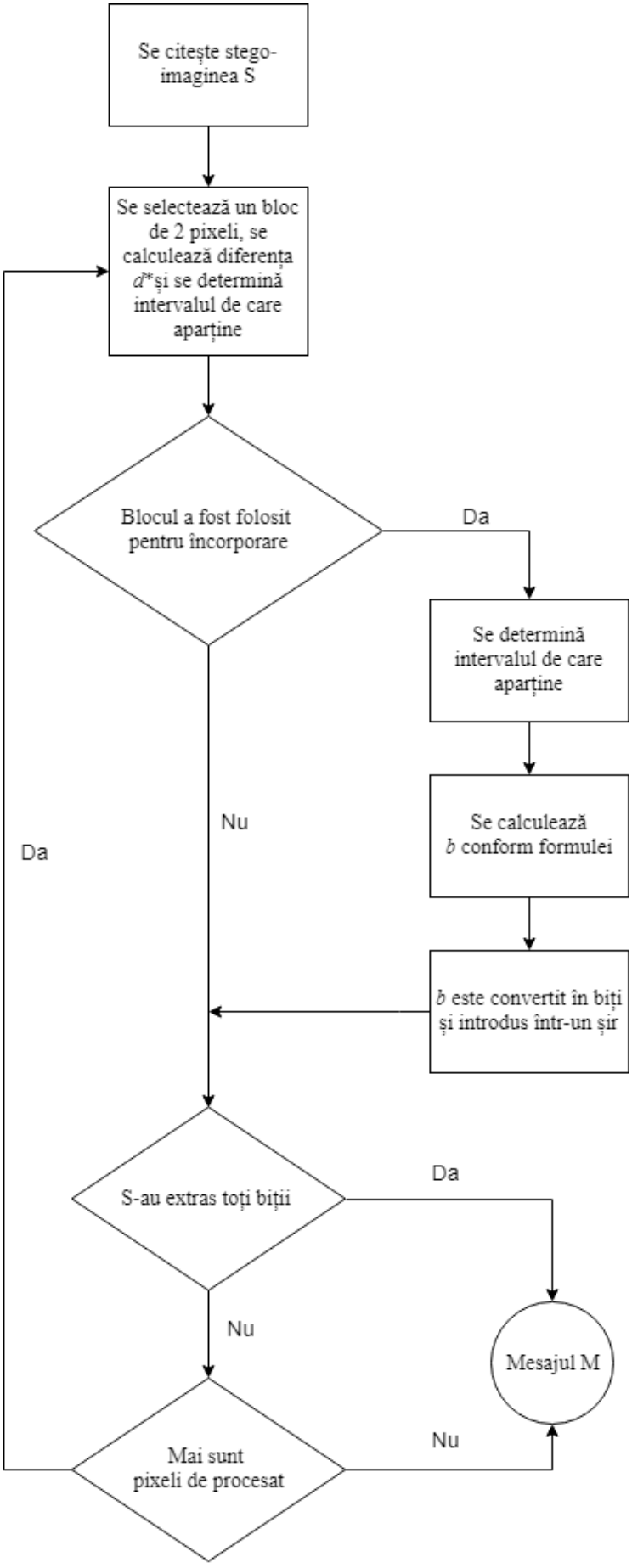

Figura 5. Extragerea mesajului

Algoritmul în pseudocod:

Input: stego-imaginea S

Output: mesajul M

```
//se extrage lungimea mesajului întâi
...
...
messageLength = toValue(messageLengthBits);
messageLength *= 8;
messageBits;
i = 0, j = 0;
while(i < S.rows && messageBits.size() < messageLength){
       while(j < S.cols && messageBits.size() < messageLength){
              p1 = S.at(i, j);
              p2 = S.at(i, j+1);
              d* = p1 – p2;
              interval = getInterval(d*);
              check1, check2 = computePixelValues(p1, p2, d* -
                                                  interval.u);
              if(check1 < 0 || check1 > 255
              || check2 < 0 || check2 > 255){
                     j+=2;
                     continue;
              }

              b = getEmbeddedValue(d*, interval.lk);
              bits = toBinary(b);
              messageBits.append(b);
              j+=2;
              }
       j = 0;
       i++;
}
return toMessage(messageBits);
```

2.2 Rezultate și observații:

Pentru analiza algoritmului, mulțimea intervalelor de cuantificare utilizată este

$M = \{ [0,7], [8,15], [16,31], [32,63], [64,127], [128,255] \}$.

Tabel 3 – Capacitățile de încorporare ale imaginii utilizate

Imagine purtător	Dimensiune	Capacitate maximă (după intervalele de cuantificare)
dog_256x256.png	256 * 256	79,474

Textul utilizat în analiza LSB, conform calculelor, este prea mare (dimensiunea de 129,640 biți > capacitatea maximă). În ciuda acestui fapt, îl vom folosi în continuare, pentru comparație.

Fig 6. Imaginea purtătoare

Fig 6.1. Stego-imaginea

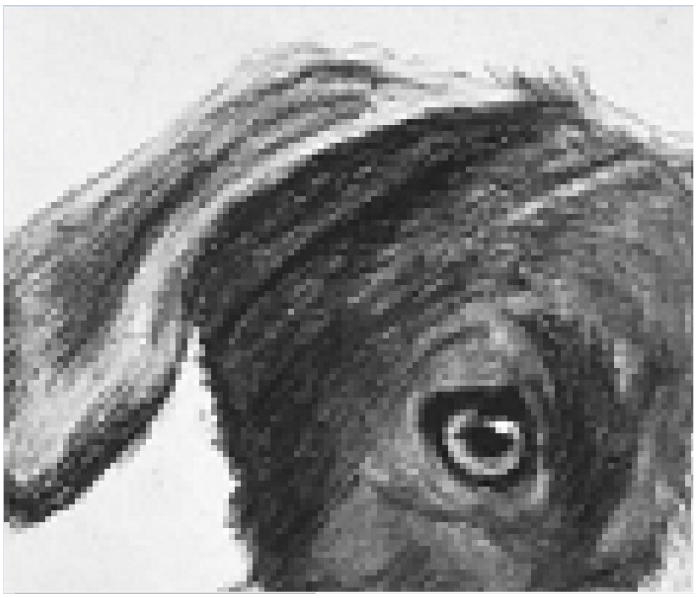

Fig 7. Imaginea purtătoare (zoom in)

Fig 7.1. Stego-imaginea (zoom in)

La prima vedere, imaginile nu au nici o diferență. Figurile 6 și 6.1 ilustrează acest fapt. Un zoom in asupra colțului din stânga sus (figurile 7, 7.1) demonstrează însă că imaginile diferă, dar informația este încorporată diferit în funcție de uniformitate zonei.

Pentru a introduce cât mai mult din mesaj, putem modifica intervalele de cuantizare astfel încât imaginea să aibă o capacitate mai mare. Intervalele devin astfel:

$$M' = \left\{ \begin{matrix} [0,15],[16,23],[24,31],[32,47],[48,63],[64,79],[80,95],[96,103],[104,111], \\ [112,119],[120,127],[128,135],[136,143], \\ [144,151],[152,159],[160,191],[192,255], \end{matrix} \right\}$$

Noua capacitate devine astfel 112,149 biți. Rezultatele sunt vizibile în figurile 8 și 8.1. Figurile 9 și 9.1 arată diferențele la nivel mărit.

Fig 8. Imaginea purtătoare

Fig 8.1. Stego-imaginea

Fig 9. Imaginea purtătoare (zoom in)

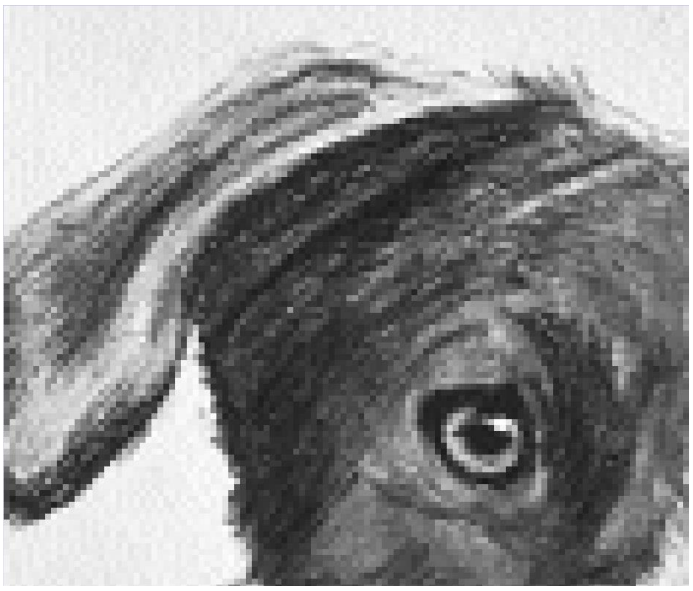

Fig 9.1. Stego-imaginea (zoom in)

Modificările din stego-imagine sunt vizibile doar atunci când cadrul este mărit. Zgomotul însă este distribuit uniform, iar în absența imaginii originale, o persoană obișnuită nu își poate da seama că imaginea ascunde un mesaj.

Tabel 4 – Metricile obținute pentru PVD, în ordine, pentru fiecare mulțime de intervale

Mulțime	MSE	PSNR
M	0.00012765242718160152	87.0705108642578125
M’	0.0001240782439708709 7	87.19384765625

Tabelul 4 ilustrează rezultatele obținute din punctul de vedere al metricilor. Rata de eroare și rata semnal-zgomot sunt destul de apropiate ca valoare, însă în cazul *M’*, aproximativ 90% din mesaj a fost introdus. Valorile sunt mai bune pentru *M’* deoarece imaginea are multe zone neuniforme, pentru care s-a ales să se introducă un număr mai mic de biți/bloc.

Concluzia este așadar că algoritmul este robust. Mesajul este ascuns astfel încât stego-imaginea rezultată să nu aibă diferențe observabile de ochiul uman, folosindu-se de zonele neuniforme. Mulțimea de intervale poate fi aleasă în funcție de imaginea purtătoare, de rezultatul dorit și de lungimea mesajului. ”Algoritmul nu oferă numai o modalitate mai bună de a încorpora o un număr mare de date, ci asigură și o cale de a obține confidențialitate.”[4]

3. EDS (**Ed**ge based **S**teganography):

Acest algoritm propune ascunderea mesajului în muchiile imaginii. Sunt alese muchiile deoarece ochiul uman nu este suficient de sensibil la modificările aduse în zone neuniforme.

3.1 Algoritmul:

Schema se folosește de algoritmul lui Canny de detectare a muchiilor, ele fiind selectate pe baza lungimii mesajului. O stego-cheie este generată în timpul algoritmului, pixelii muchiilor fiind permutați în funcție de aceasta, pentru a nu introduce biții în pixeli în ordinea apariției lor în muchii.

3.1.1 Încorporarea:

Este important să menționăm că schimbarea biților din imaginea purtător poate determina detectarea unor muchii greșite din stego-imagine în momentul extragerii. ”Pentru a păstra muchiile înainte și după încorporare, biții LSB ai imaginii purtător sunt mascați, iar detectarea muchiilor este aplicată pe imaginea mascată. Deoarece LSB(1) nu modifică decât LSB ai imaginii purtător, muchiile imaginii purtător și ale stego-imaginii rămân identice.”[5]

Pentru a obține muchiile, algoritmul lui Canny este utilizat. Selectarea muchiilor se face pe baza lungimii mesajului și a imaginii. Acest algoritm se folosește de un prag pentru a determina muchiile. Un prag mai mare înseamnă mai multe muchii. Așadar, cu cât lungimea mesajului este mai mare, cu atât pragul este mai mare, pentru a obține cât mai multe muchii în care să fie introdus mesajul.

Parametrii algoritmului lui Canny sunt pragul superior (t_h), pragul inferior (t_l) și lățimea matricii Gaussiene (w). Pragul superior este utilizat pentru detectarea muchiilor principale, iar pragul inferior este folosit pentru detectarea muchiilor adiacente. Sensibilitatea algoritmului la zgomot este dată de lățimea matricii.

Pragul superior t_h este calculat în momentul executării algoritmului. Considerăm inițial pragul minim $t_{min} = 0$ și pragul maxim $t_{max} = 1000$. t_h se află în acest interval, și se

calculează prin căutare binară. Se determină numărul de muchii pentru pragul median $\frac{t_{max}+ t_{min}}{2}$. Se verifică dacă numărul de pixeli ai muchiilor returnate n_e este suficient pentru a ascunde mesajul. Deoarece este posibil ca n_e să nu fie de aceeași lungime ca a mesajului, se impune ca n_e să fie mai mare decât numărul de pixeli N. De asemenea, se setează o limită superioară L pentru n_e. În acest algoritm, $L = 0,1 * N$. Pragul inferior t_l este $0,4 * t_h$.

Dacă $n_e < N$, atunci t_{min} = pragul median, iar dacă $n_e > N + L$, atunci t_{max} = pragul median, până când $N > n_e > N + L$.

Algoritmul de obținere al pragului:

Input: imaginea *I*, lungimea mesajului *N*, lățimea matricii Gaussiene *w*

Output: pragul t_h

```
L = 0,1 * N;
tmax = 1000;
tmin = 0;
set = false
repeat{
       th = (tmax+ tmin)/2;
       edgeImage = Canny(I, th, 0,4*th, w);
       ne = getEdgePixelCount(edgeImage);
       difference = ne – N;
       if(difference > L){
              tmin = th;
       }
       else if(difference < 0){
              tmax = th;
       }
       else set = true
}until set = true;
return th;
```

Odată obținut pragul, se calculează muchiile în care se va introduce mesajul. Cu ajutorul stego-cheii, pixelii muchiilor sunt permutați, iar mesajul este încorporat în LSB mascați anterior.

Pentru extragere, receptorul are nevoie de pragul calculat și de lățimea utilizată în algoritmul lui Canny. Acestea două sunt încorporate în pixelii ce nu aparțin de muchii. Se aplică algoritmul lui Canny pe imaginea mascată, pragurile fiind egale cu 0 iar lățimea egală cu 3 (pentru a obține toate muchiile posibile). Sunt luați în ordine pixelii complementari și în ei se ascund pragul și lățimea.

Figura 10 ilustrează, pentru un mesaj cu lungimea de 873 caractere, muchiile disponibile pentru încorporare, și pixelii în care s-a încorporat.

Figura 10. În ordine, imaginea purtătoare, imaginea cu muchiile disponibile, imaginea cu muchiile în care s-a încorporat

Figura 11 ilustrează pașii algoritmului.

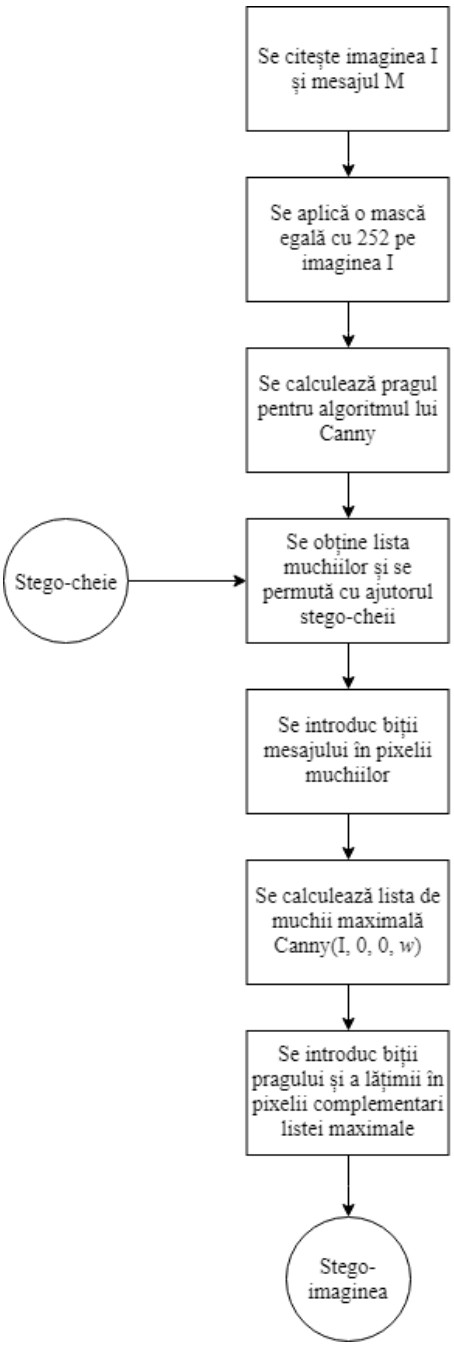

Figura 11. Încorporarea mesajului

Algoritmul în pseudocod:

Input: Imaginea I, mesajul M, lățimea *w*

Output: Stego-imaginea S

```
S = I;
messageLength = |M|;
secretBits = toBits(messageLength) +  toBits(M);
i = 0;
I = bitwise_and(I, 252);
t_h = getThreshold(I, messageLength, w);
edgeImage = Canny(I, t_h, t_h * 0,4, w);

//din edgeImage obținem o listă cu pixelii muchiilor
edgeMap = toEdgeMap(edgeImage);

stegoKey = getStegoKey();
edgeMapPermutation(edgeMap, stegoKey);

bitIndex = 0;
while(i < edgeMap.size() && bitIndex <
                                secretBits.size()){
      //coordonatele pixelului
      x, y = edgeMap[i];
      pixel = S.at(x, y);
      pixelBits = toBits(pixel);
      pixelBits[0] = secretBits[bitIndex++];
      pixelBits[1] = secretBits[bitIndex++];

      S.at(x, y) = toPixel(pixelBits);
      i++;
}

//șiruri de 32 de biți
thresholdBits = toBits(t_h);
widthBits = toBits(w);

maxEdgeImage = Canny(I, 0, 0, 3);
i = 0, j = 0, index = 0;
for(i = 0; i < S.rows && index < 32; i++){
      for(j = 0; j < S.cols && index < 32; j++){
            pixelCheck = maxEdgeImage.at(i, j);
            if(pixelCheck == 0){
                  pixel = S.at(i, j);
                  pixelBits = toBits(pixel);
                  pixelBits[0] = thresholdBits[index++];
                  S.at(i, j) = toPixel(pixelBits);
            }
      }
}

//analog pentru widthBits
return S;
```

3.1.2 Extragerea:

Extragerea mesajului se face prin obținerea pragului și a lățimii utilizate anterior, prin procesul invers. Asupra stego-imaginii se aplică aceeași mască pentru a obține aceleași muchii. Se calculează toate muchiile posibile, aplicând algoritmul lui Canny cu pragurile 0 și lățimea matricii 3, urmând ca pe baza rezultatului să se extragă pragul t_h și lățimea w folosite în încorporare. Pentru extragerea mesajului propriu-zis, se obțin muchiile folosind t_h și w obținute anterior. Stego-cheia generată trebuie transmisă prin alt mediu însă. Muchiile generate sunt permutate conform stego-cheii, iar biții lungimii mesajului și ai mesajului se extrag.
Figura 12 ilustrează pașii.

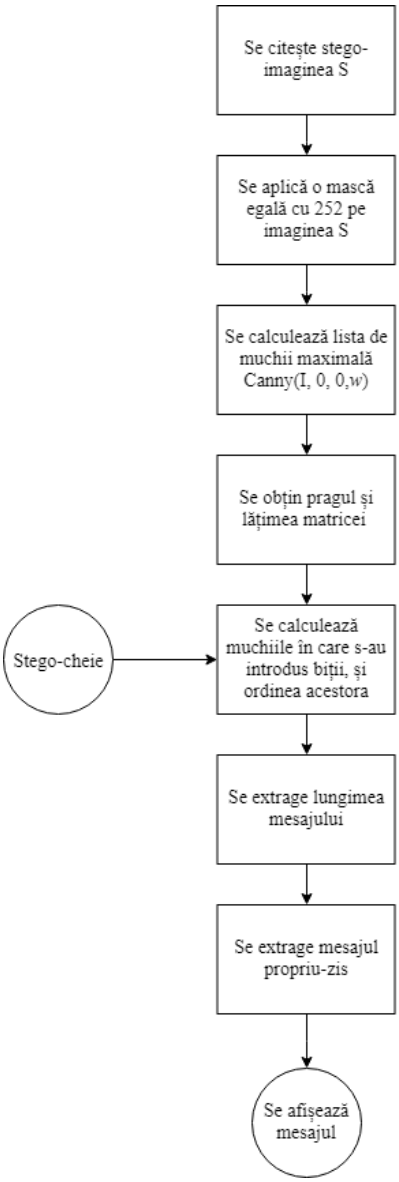

Figura 12. Extragerea mesajului

Algoritmul în pseudocod:

Input: Stego-imaginea S, stego-cheia stegoKey

Output: Mesajul M

```
S' = S;
S' = bitwise_and(252);
maxEdgeImage = Canny(S', 0, 0, 3);
for(i = 0; i < S.rows && index < 32; i++){
       for(j = 0; j < S.cols && index < 32; j++)
              pixelCheck = maxEdgeImage.at(i, j);
              if(pixelCheck == 0){
                     pixel = S.at(i, j);
                     pixelBits = toBits(pixel);
                     thresholdBits[index++] =
                                   pixelBits[0];
              }
       }
}
t_h = toValue(thresholdBits);
//analog pentru width

edgeImage = Canny(S', t_h, t_h * 0,4, w);
edgeMap = toEdgeMap(edgeImage);
edgeMapPermutation(edgeMap, stegoKey);

i = 0, j = 0, index = 0;
while(i < edgeMap.size() && index < 16){
       //coordonatele pixelului
       x, y = edgeMap[i];
       pixel = S.at(x, y);
              pixelBits = toBits(pixel);
              messageLengthBits[index++] = pixelBits[0];
              messageLengthBits[index++] = pixelBits[1];
              i++;
}
messageLength = toValue(messageLengthBits);
messageLength *= 8;
index = 0;
while(i < edgeMap.size() && index < messageLength){
              //coordonatele pixelului
              x, y = edgeMap[i];
              pixel = S.at(x, y);
              pixelBits = toBits(pixel);
              messageBits[index++] = pixelBits[0];
              messageBits[index++] = pixelBits[1];
              i++;
}
return toMessage(messageBits);
```

3.2 Rezultate şi observaţii:

Pentru analiza algoritmului, s-a ales ca lățimea matricii Gaussiene să fie 3 (algoritmul lui Canny acceptă doar lățimi impare). Deoarece algoritmul ascunde biții în LSB ai pixelilor muchiei, este necesar ca textul să fie unul mai scurt. Din acest motiv, au fost alese drept mesaj câteva paragrafe dintr-un articol[6]. Textul are 2022 de caractere (16,176 biți).

Tabel 5 – Capacitatea imaginii folosite raportată la numărul maxim de muchii

Imaginea purtător	Dimensiune	Capacitate
dog_256x256.png	256 * 256	42,998

Pentru a obține o stego-cheie aleatoare, am ales ca aceasta să fie citită de la tastatură. În contextul analizei, stego-cheia folosită va fi ”4642”.

Figura 13. Imaginea originală

Figura 13.1 Stego-imaginea

Figura 13.2 Pixelii muchiilor disponibili pentru încorporare

Figura 13.3 Pixelii utilizați după permutare

Deoarece algoritmul ascunde informația în muchiile imaginii, diferențele sunt aproape invizibile pentru ochiul uman.

Tabel 6 – Metricile obținute pentru EDS

MSE	PSNR
0.00000433670356869698	101.7592010498046875

Rata de eroare indică faptul că imaginile au grad de similaritate foarte ridicat, iar raportul semnal-zgomot ridicat sugerează că zgomotul este aproape absent.

În concluzie, algoritmul este unul sigur. Informația este este ascunsă astfel încât prezența ei să fie aproape imperceptibilă de ochiul uman. Utilizarea stego-cheii oferă un grad de securitate ridicat, modificând ordinea normală în care este încorporată informația. Pe de altă parte, algoritmul nu acceptă cantități mari de informații precum PVD și LSB. De aceea, este necesară alegerea unei imagini potrivite pentru mesajul ce trebuie trimis.

4. DCT (**D**iscrete **C**osine **T**ransform based steganography):

"Cele mai populare tehnici de ascundere a mesajelor sunt tehnicile steganografice ce se bazează pe domeniul spațial și pe domeniul transformatelor. […] În domeniul transformatelor, imaginea purtătoare este transformată în domeniul de frecvențe, cu ajutorul transformatei Fourier Discrete, a transformatei Cosinus Discrete (DCT), etc. […] În DCT, biții LSB ai coeficienților DCT ai imaginii purtătoare sunt înlocuiți cu biții mesajului."[7]

4.1 Algoritmul:

Algoritmul propune modificarea imaginii la nivelul transformatei Cosinus Discrete. Imaginea este parcursă iterând prin blocuri de dimensiune 8 * 8 asupra cărora se aplică DCT. DCT se calculează astfel:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\left[\frac{\pi(2x+1)u}{16}\right]\cos\left[\frac{\pi(2y+1)v}{16}\right]$$

unde

$$u = \overline{0,7}, v = \overline{0,7}, f(x,y) = valoarea\ pixelului\ din\ blocul\ curent$$

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & altfel \end{cases}$$

După încorporarea biților mesajului în coeficienții DCT, se calculează transformata inversă (IDCT) pentru a obține blocul stego-imaginii cu pixelii modificați conform noilor coeficienți. IDCT se calculează astfel:

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\left[\frac{\pi(2x+1)u}{16}\right]\cos\left[\frac{\pi(2y+1)v}{16}\right]$$

$x = \overline{0,7}, y = \overline{0,7}$

Trebuie menționat că extragerea din algoritm nu este perfectă. Modificarea unui coeficient determină modificarea și a celorlalți coeficienți într-o oarecare măsură. Din acest motiv, există șanse mari ca mesajul extras să fie alterat. De aceea, algoritmul nu încorporează și lungimea mesajului. Aceasta trebuie trimisă printr-un alt mediu.

4.1.1 Încorporarea:

Pentru încorporare, imaginea este împărțită în blocuri de dimensiune 8 * 8, asupra cărora se aplică DCT. Odată obținută matricea coeficienților, se modifică elementul de pe poziția 0, 0, deoarece acesta are asupra tuturor celorlalți coeficienți, atunci când se inversează procesul transformatei pentru a obține blocul modificat.

Deoarece coeficienții sunt numere reale, aceștia sunt stocați în variabile de tip *float*, fiind pe 32 biți. Pentru a nu modifica partea întreagă a coeficientului, biții mesajului sunt încorporați în biții mantisei. În implementare, s-a ales încorporarea a 4 biți pe coeficient. Experimental, am determinat că biții cei mai potriviți pentru a fi schimbați sunt biții de pe pozițiile 6 – 9.

După încorporare, se aplică IDCT asupra matricii de coeficienți, și se obține stego-blocul propriu-zis. Acesta înlocuiește blocul original în stego-imagine. Procesul se repetă până când mesajul a fost încorporat complet, sau imaginea nu mai permite încorporare.

Figura 13 ilustrează pașii încorporării.

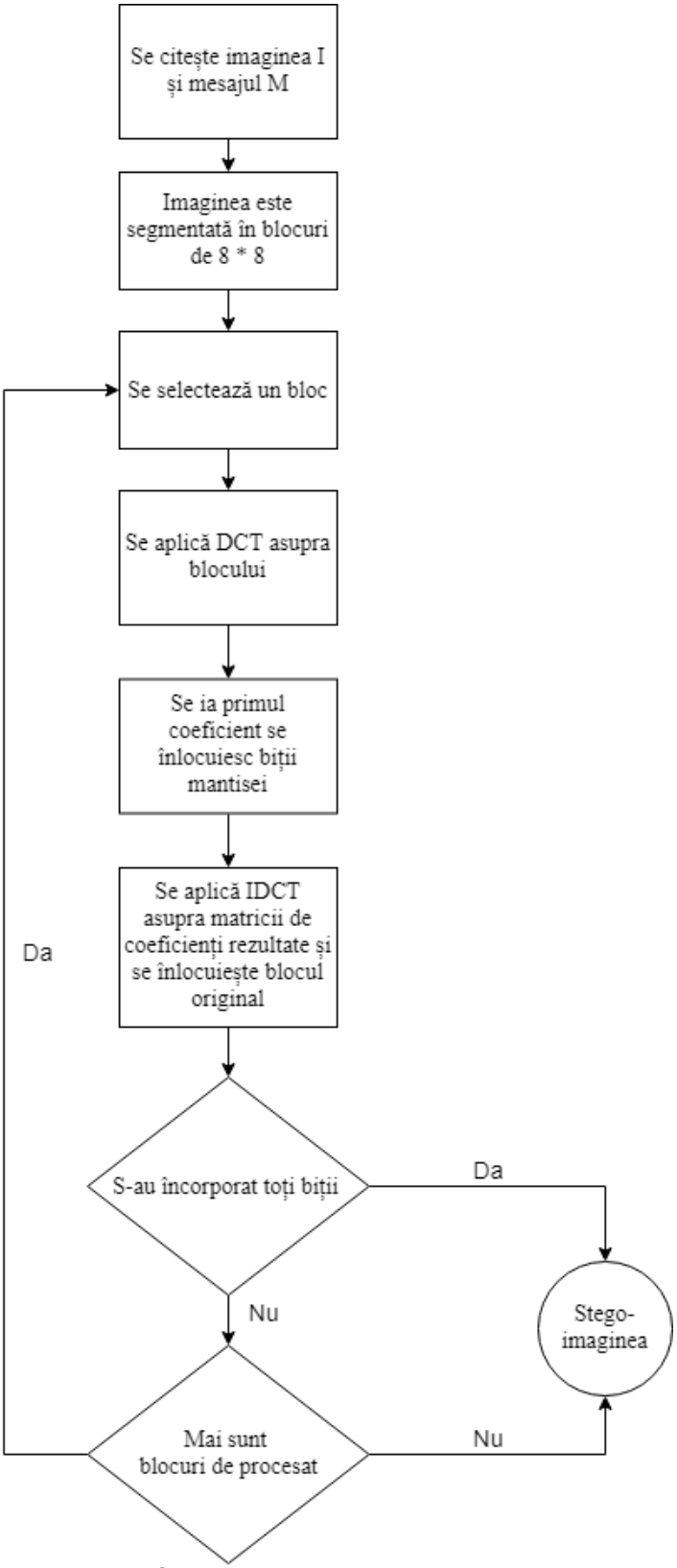

Figura 13. Încorporarea mesajului

Algoritmul în pseudocod:

Input: imaginea purtător, mesajul M

Output: stego-imaginea

```
S = I;
secretBits = toBits(messageLength) +  toBits(M);

//obținem ultimul multiplu de 8 mai mic decât numărul de
coloane/linii
leftEdge = getLeftEdge(I.cols);
botEdge = getBotEdge(I.rows);

bitIndex = 0, i = 0, j = 0;
while(i < botEdge && bitIndex < secretBits.size()) {
  while(j < leftEdge && bitIndex < secretBits.size()) {
    block = getBlock(S, i, j);
    dctBlock = DCT(block);
    pixelVal = dctBlock.at(0, 0);
    if (pixelVal != 0) {
      pixelBits = toBits(pixelVal);
      ind = 14;
      while(ind < 18  & bitIndex < secretBits.size()){
        pixelBits[ind++] = secretBits[bitIndex++];
      }
      dctBlock.at(0, 0) = toValue(pixelBits);
    }
    block = IDCT(dctBlock);
    replaceBlock(S, block, i, j);
    j++;
  }
  i++;
}
return S;
```

4.1.2 Extragerea:

Extragerea mesajului se face prin același proces de segmentare a imaginii purtătoare în blocuri de dimensiune 8 * 8. Blocurile sunt parcurse în aceeași ordine, aplicându-se DCT pentru a obține matricea de coeficienți, din care se extrag biții de pe pozițiile în care s-a încorporat anterior, până când mesajul este extras complet sau nu mai sunt blocuri de procesat. Figura 14 ilustrează pașii extragerii.

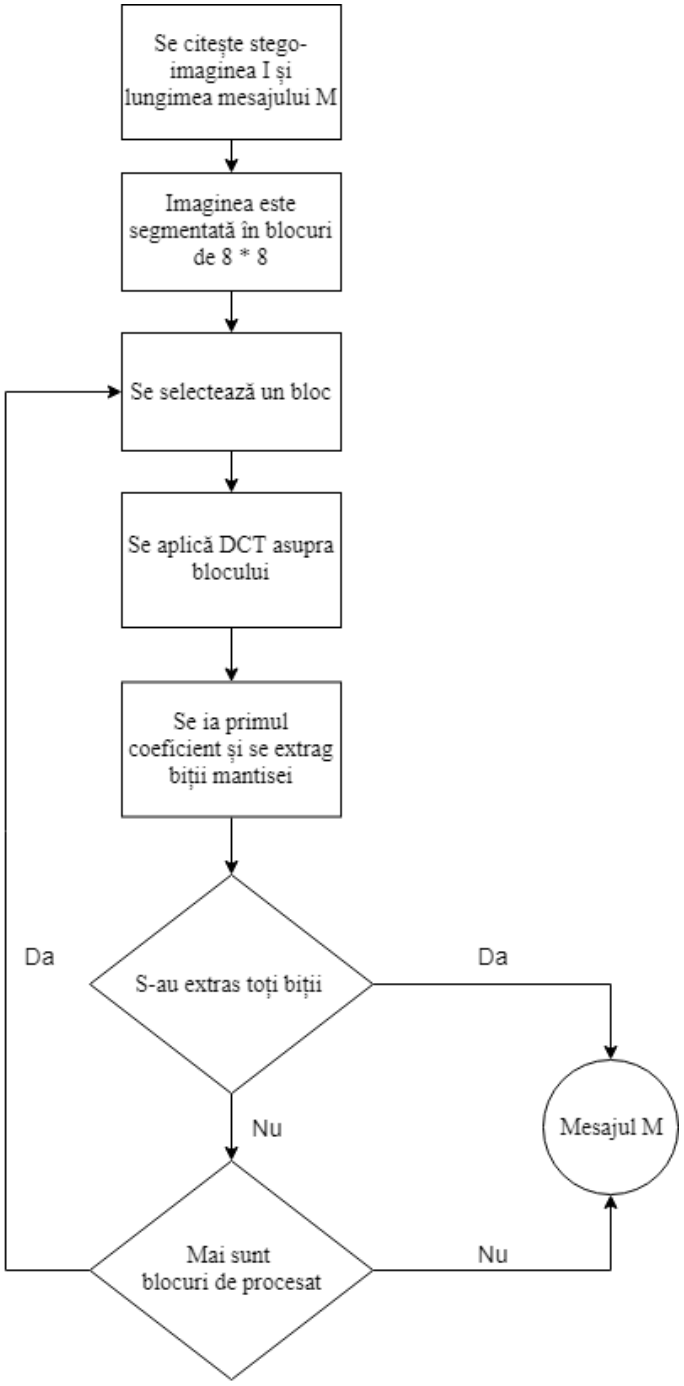

Figura 14. Extragerea mesajului

Algoritmul în pseudocod:

Input: stego-imaginea S, lungimea mesajului M

Output: mesajul M

```
S;

//obținem ultimul multiplu de 8 mai mic decât numărul de
coloane/linii
leftEdge = getLeftEdge(s.cols);
botEdge = getBotEdge(s.rows);

bitIndex = 0, i = 0, j = 0;
while(i < botEdge && bitIndex < messageLength) {
  while(j < leftEdge && bitIndex < messageLength) {
    block = getBlock(S, i, j);
    dctBlock  = DCT(block);
    pixelVal = dctBlock.at(0, 0);
    if (pixelVal != 0) {
      pixelBits = toBits(pixelVal);
      ind = 14;
      while(ind < 18  & bitIndex < messageLength){
        secretBits.append(pixelBits[ind++]);
      }
    }
    j++;
  }
  i++;
}
return S;
```

4.2 Rezultate și observații:

Pentru analiza algoritmului am utilizat același text de 2022 de caractere.

Tabel 7 – Capacitatea imaginii folosite

Imaginea purtător	Dimensiune	Capacitate
dog.png	768 * 512	23,896

Figura 15. Imaginea purtătoare

Figura 15.1. Stego-imaginea

Figurile 15 și 15.1 ilustrează faptul că imaginile sunt imposibil de diferențiat pentru ochiul uman.

Tabel 8 – Metricile obținute pentru DCT

MSE	PSNR
0.00000000004207683160	151.890380859375

Rata de eroare pentru DCT este extrem de mică, raportul semnal-zgomot fiind în schimb foarte mare, comparativ cu ceilalți algoritmi. Dezavantajul algoritmului însă este faptul că, deși imaginea are dimensiuni mult mai mari decât cea folosită anterior (”dog_256x256.png”), capacitatea ei este foarte mică. Putem mări capacitatea folosind mai mulți coeficienți. În următoarea iterație a algoritmului, coeficienții utilizați sunt cei de deasupra diagonalei principale, crescând astfel capacitatea la 835,656 biți.

Dezavantajul în această iterație este dat de faptul că mesajul nu este extras în întregime corect. Figura 16 ilustrează mesajul după extragere.

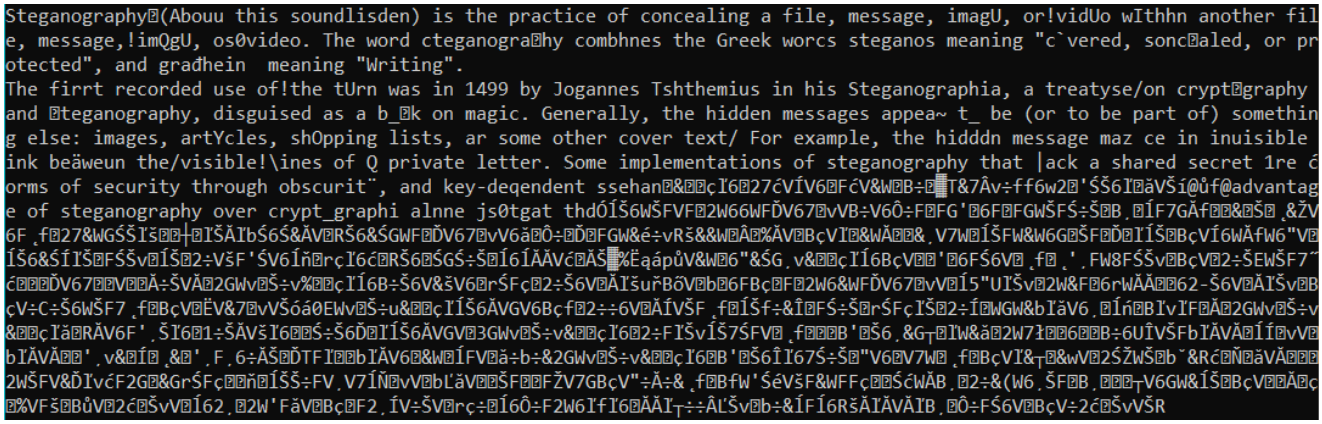

Figura 16. Mesajul extras

După cum se observă, prima jumătate este parțial lizibilă, însă a doua jumătate nu este inteligilă. Metricile indică rezultate bune, dar mesajul afișat nu este de folos.

Tabel 8.1

MSE	PSNR
0.00000000000002227190	184.653228759765625

Pe de altă parte, am determinat experimental că această iterație este mai potrivită pentru ascunderea unei alte imagini.

Imaginea ascunsă poate fi, spre exemplu, o imagine ce conține un mesaj criptat.

Figura 17. Imaginea cu mesajul criptat

Figura 17.1. Imaginea primită de receptor

Tabel 8.2

MSE	PSNR
0.0000000000025280715	164.1029205322265625

Rezultatul de această dată este inteligibil, iar metricile indică modificări insesizabile.

În concluzie, algoritmul utilizează transformata Cosinus Discretă pentru a ascunde în blocuri de dimensiune 8 * 8, în mod uniform, informația ce trebuie transmisă. El poate fi adaptat pentru a ascunde cantități mari de date sub forma unor imagini, fără modificări vizibile.

Concluzii

În această lucrare am analizat steganografia și câteva din conceptele ei de bază. Am oferit de asemenea și o prezentare generală asupra 4 algoritmi ce pot fi utilizați pentru comunicare confidențială, și am analizat performanțele acestora raportându-ne la calitatea stego-imaginii rezultate.

Direcția pe viitor a steganografiei, așa cum este menționat în [8], este selectarea adaptivă a locațiilor în care se încorporează datele. Sunt dezvoltate deja metode steganografice ce folosesc o strategie de încorporare adaptivă pentru a utiliza zonele neuniforme, pentru a evita crearea unor artefacte perceptibile. Sunt greu de construit modele statistice asupra zonelor cu muchii sau neuniforme, determinând luarea unor decizii false de către schemele steganalitice.

Bibliografie

[1] – Ingemar J. Cox, Matthew L. Miller, Jeffrey A. Bloom, Jessica Fridrich, and Ton Kalker - *Digital Watermarking and Steganography, Second Edition*

[2] – https://edition.cnn.com/2012/04/30/world/al-qaeda-documents-future/index.html

[3] – https://opencv.org/

[4] – Da-Chun Wu, Wen-Hsiang Tsai. *A steganographic method for images by pixel-value differencing* – Pattern Recognition Letters 24 (2003)

[5] – Saiful Islam, Mangat R. Modi, Phalguni Gupta. *Edge-based image steganography* – EURASIP Journal on Information Security 2014

[6] – https://en.wikipedia.org/wiki/Steganography

[7] – K. B. Shiva Kumar, K. B. Raja, R. K. Chhotaray, Sabyasachi Pattanaik – *Bit Length Replacement Steganography based on DCT Coefficients* – International Journal of Engineering Science and Technology

[8] – B. Li, J.H. He, J.W. Huang, and Y.Q. Shi – *A Survey on Image Steganography and Steganalysis* – Journal of Information Hiding and Multimedia Signal Processing