Proiect la Programare Procedurala

Modulul de criptare/decriptare

Functia XORSHIFT32(int numbers,unsigned int *RANDOM,unsigned int r) unde:

- valoarea variabilei numbers este 2*numarul de pixeli din poza fiind de tipul int
- vectorul RANDOM retine numerele care sunt generate fiind de dimensiune numbers si acesta este de tipul unsigned int
- R0 este valoare R0 citita din secret_key.txt care este prima valoare a vectorului RANDOM fiind de tipul unsigned int

Functia XORSHIFT32 este implementarea generatorului de numere pseudo aleatoare, propus de George Marsaglia în 2003 generează numere întregi fără semn pe 32 de biți, cu un caracter pseudo-aleator foarte bun, folosind operații de deplasare pe biți și XOR.

```
unsigned int r, seed, k;
.....
r = seed;
for(k = 0; k < n; k++)
{
    r = r ^ r << 13;
    r = r ^ r >> 17;
    r = r ^ r << 5;
    printf("%u\n", r);
}
```

```
seed = 1000;

266172694
3204629577
385443340
2099747088
1321081938
733430728
3121244111
680290934
.....
```

Functia unsigned char *liniarizare_poza(char* nume_fisier_sursa,unsigned char *header,unsigned int *dim_img,unsigned int *inaltime_img,unsigned int *latime_img, int *padding) unde:

- nume_fisier_sursa este denumirea pozei BMP
- vectorul header reprezinta header-ul pozei BMP
- dim_img este dimensiunea imaginii in octeti
- inaltime_img este numarul de linii pe care le are poza
- latime_img este numarul de coloane pe care le are poza
- padding sunt octetii complectati pentru a respecta formatul de numar divizibil cu 4 pe linie

In functie se deschide poza, daca aceasta nu este gasita se afiseaza mesajul “nu am gasit imaginea cautata”, urmand sa se citeasca dim_img, latime_img, inaltime_img , calculam padding-ul, urmand sa tinem in memorie poza citita sub forma liniarizata unde un pixel poate fi interpretat ca 3 valori consecutive in vector incepand de la pozitia i*latime_img*3+j*3 pentru un pixel de pe linia i si coloana j, deoarece liniile sunt rasturnate se actualizeaza pixelii in ordine de la pozitia (inaltime_img-i-1)*latime_img*3+j*3. Functia returneaza vectorul format Q care retine pixelii imaginii liniarizate.

Functia are rolul de a liniariza o poza.

Functia void afisare(char *nume_fisier_iesire,unsigned char *Poza,unsigned char *header,unsigned int dim_img,unsigned int inaltime_img,unsigned int latime_img,int padding) unde:

- nume_fisier_iesire este numele pozei pe care dorim sa o afisam
- in vectorul Poza sunt pixelii pozei retinuti in forma liniarizata
- vectorul header reprezinta header-ul pozei
- dim_img este dimensiunea imaginii in octeti
- inaltime_img este numarul de linii pe care le are poza
- latime_img este numarul de coloane pe care le are poza
- padding sunt octetii complectati pentru a respecta formatul de numar divizibil cu 4 pe linie

Functia are rolul de a afisa o poza. In interiorul functiei se afiseaza header-ul apoi urmat de liniile matricei pozei care sunt rasturnate din nou pentru a nu iesi poza invers astfel se afiseaza liniile in ordine inversa pentru a nu intampina aceasta problema.

Functia void permut_pixeli_poza(unsigned char *A,unsigned char *B,unsigned int *P,unsigned int inaltime_img,unsigned int latime_img) unde:

- vectorul A tine valorile pixelilor, pozei B, permutati
- vectorul B retine valorile pixelilor pozei
- vectorul P este permutarea dupa care se permuta imaginea B
- inaltime_img este inaltime imaginii
- latime_img este latimea imaginii

Functia are rolul de a permuta pixelii din B dupa o permutare P si o retine in A.

In interiorul functiei se permuta pixelii dupa regula A[P[i]]=B[i] respectand indicii aferenti(noul pixel P[i] ia valoarea pixelului i).

Functia void read_secret_key(char *nume_cheie,unsigned int *R0,unsigned int *SV) unde:

- nume_cheie este numele fisirului unde se gasesc valorile cerute
- R0 este seed-ul pentru generatorul de numere random (XORSHIFT32)
- SV (starting value) este un număr întreg nenul fără semn pe 32 de biți

Functia are rolul de a citi valorile pentru variabilele R0 si SV.

Functia unsigned int *fac_permutarea(char *nume_cheie,unsigned int *SV,unsigned int inaltime_img,unsigned int latime_img,unsigned int Random[]) unde:

- nume_cheie este numele pentru accesul la R0 si SV
- SV (starting value) este un număr întreg nenul fără semn pe 32 de biți
- inaltime_img este inaltimea imaginii
- latime_img este latime imaginii
- Random este vectorul unde se tin valorile random create in functia XORSHIFT32

Functia are rolul de a generea permutarea aleatoare P de dimensiune numar de pixeli , folosind algoritmul lui Durstenfeld și numerele pseudo-aleatoare din vectorul Random (incepand cu indexul 1).

In interiorul functiei se apeleaza functia read_secret_key pentru a se citi valorile pentru R0 si SV urmate de apelarea functiei XORSHIFT32 pentru a se genera sirul de numere de lungime 2*nr_de_pixeli , urmata de aplicarea propriu zisa a algoritmului lui Durstenfeld.

```
unsigned int r, n, k, p[100];
.....
for(k = 0; k < n; k++)
    p[k] = k;

for(k = n-1; k >= 1; k--)
{
    r = random(0, k);
    aux = p[r];
    p[r] = p[k];
    p[k] = aux;
}
```

n = 6

k	r	p
–	–	1 2 3 4 5 6
5	3	1 2 3 **6** 5 **4**
4	1	1 **5** 3 6 **2** 4
3	1	1 **6** 3 **5** 2 4
2	2	1 6 **3** 5 2 4
1	0	**6** **1** 3 5 2 4

unde prin `random(0,k)` am notat un număr aleator cuprins între 0 și k.

Functia void criptare_poza(char *nume_cheie,unsigned char *Poza,unsigned int inaltime_img,unsigned int latime_img) unde:

- nume_cheie este numele pentru accesul la R0 si SV
- vectorul Poza retine o Poza liniarizata
- inaltime_img este inaltimea imaginii
- latime_img este latimea imaginii

Functia are rolul de a cripta poza.

In interiorul functiei se apeleaza functia fac_permutarea pentru a se genera permutarea dupa care se permuta poza conform permutarii apeland functia permut_pixeli_poza astfel poza nou generata se afla in PozaPermutata dupa care se codifica imaginea conform pozei

d) imaginea criptată $C = (C_0, C_1, \dots, C_{W*H-1})$ (*ciphered image*) se obține aplicând asupra fiecărui pixel al imaginii $P' = (P'_0, P'_1, \dots, P'_{W*H-1})$ următoarea relație de substituție:

$$C_k = \begin{cases} SV \oplus P'_0 \oplus R_{W*H}, & k = 0 \\ C_{k-1} \oplus P'_k \oplus R_{W*H+k}, & k \in \{1,2,\dots,W*H-1\} \end{cases}$$

unde SV (*starting value*) este un număr întreg nenul fără semn pe 32 de biți.

```
/// codific poza
unsigned int act = SV;
int ind = inaltime_img*latime_img;
int i,j;
for(i = 0; i < inaltime_img; i++)
    for(j = 0; j < latime_img; j++)
    {
        act^=Random[ind];
        act^=PozaPermutata[i*latime_img*3+j*3];
        act^=((unsigned int)PozaPermutata[i*latime_img*3+j*3+1]<<8);
        act^=((unsigned int)PozaPermutata[i*latime_img*3+j*3+2]<<16);
        Poza[i*latime_img*3+j*3]   =  act     &((1<<8)-1);
        Poza[i*latime_img*3+j*3+1] = (act>>8) &((1<<8)-1);
        Poza[i*latime_img*3+j*3+2] = (act>>16)&((1<<8)-1);
        ind++;
    }
```

La sfarsitul functiei se dezaloca memoria folosita.

Functia void decriptare_poza(char *nume_cheie,unsigned char *Poza,unsigned int inaltime_img,unsigned int latime_img) unde:

- nume_cheie este numele pentru accesul la R0 si SV
- vectorul Poza retine o Poza liniarizata
- inaltime_img este inaltimea imaginii
- latime_img este latimea imaginii

Functia are rolul de a decripta o poza criptata.

In interiorul functiei se apeleaza functia fac_permutarea pentru a se genera permutarea dupa care se face inversa permutarii notata cu invP urmata de algoritmul de decodificare

c) se aplică asupra fiecărui pixel din imaginea criptată $C = (C_0, C_1, \dots, C_{W*H-1})$ inversa relației de substituție folosită în procesul de criptare, obținându-se o imagine intermediară $C' = (C'_0, C'_1, \dots, C'_{W*H-1})$:

$$C'_k = \begin{cases} SV \oplus C_0 \oplus R_{W*H}, & k = 0 \\ C_{k-1} \oplus C_k \oplus R_{W*H+k}, & k \in \{1,2,\dots,W*H-1\} \end{cases}$$

```
/// aplic decodificarea
unsigned char * PozaSemiDecriptata;
PozaSemiDecriptata = (unsigned char*) calloc(inaltime_img*latime_img*3,sizeof(unsigned char));

unsigned int act = SV;
int ind = inaltime_img*latime_img;
for(i = 0; i < inaltime_img; i++)
    for(j = 0; j < latime_img; j++)
    {
        act^=Random[ind];
        act^=Poza[i*latime_img*3+j*3];
        act^=((unsigned int)Poza[i*latime_img*3+j*3+1]<<8);
        act^=((unsigned int)Poza[i*latime_img*3+j*3+2]<<16);
        PozaSemiDecriptata[i*latime_img*3+j*3]   = act     &((1<<8)-1);
        PozaSemiDecriptata[i*latime_img*3+j*3+1] = (act>>8) &((1<<8)-1);
        PozaSemiDecriptata[i*latime_img*3+j*3+2] = (act>>16)&((1<<8)-1);
        act = Poza[i*latime_img*3+j*3] + ((unsigned int)Poza[i*latime_img*3+j*3+1]<<8) +((unsigned int)Poza[i*latime_img*3+j*3+2]<<16);
        ind++;
    }
```

in final apelez functia permut_pixeli_poza pentru a permuta pixelii conform permutarii invP dupa dezaloc memoria.

Functia void chi_patrat(unsigned char *Poza,unsigned int inaltime_img,unsigned int latime_img) unde:

- vectorul Poza tine valorile pixelilor pozei
- inalitime_img este inaltimea imaginii
- latime_img este latimea imaginii

Functia are rolul de a calcula uniformitatea distributiei pixelilor.

In interiorul functiei calculam frecventa fiecarei nuante de culoare RGB dupa care calculam valoarea pentru fiecare nuanta RGB in functie de formula

- Valoarea testului χ^2 corespunzătoare unei imagini de dimensiune $m \times n$ pixeli se calculează folosind formula următoare:

$$\chi^2 = \sum_{i=0}^{255} \frac{(f_i - \bar{f})^2}{\bar{f}}$$

unde f_i este frecvenţa valorii i $(0 \leq i \leq 255)$ pe un canal de culoare al imaginii, iar $\bar{f}$ este frecvenţa estimată teoretic a oricărei valori i, respectiv $\bar{f} = \frac{m \times n}{256}$.

,

afisam valorile gasite si dezaloc memoria folosita.

Functia void criptare_decriptare()

In aceasta functie se citesc din fisierul “criptare_decriptare_name_file.txt” numele pozei, numele pe care il dau pozei rezultate prin aplicarea criptarii asupra pozei, numele unei poze criptate, numele pe care il dau pozei rezultate prin aplicarea decriptarii asupra pozei criptate si numele fisierul cheie unde sunt valorile R0 si SV. Retin in vectorul Poza valorile pixelilor pozei aplicand apeland functia liniarizare_poza dupa care afisez valorile obtinute pentru testul chi_patrat apeland functia chi_patrat pentru poza initiala, criptez poza apeland functia criptare_poza si afisez poza rezultata. In continuare afisez valorile obtinute pentru testul chi_patrat apeland functia chi_patrat pentru poza criptata, decriptez poza folosind functia decriptare_poza si apoi afisez poza rezultata. La sfarsit dezaloc memoria.

Modulul de recunoaștere de pattern-uri (cifre scrise de mână)

Functia void make_image_grayscale(unsigned char *Poza,unsigned int inaltime_img,unsigned int latime_img) unde:

- vectorul Poza reprezinta pixelii unei poze
- inaltime_img reprezinta inaltimea pozei
- latime_img reprezinta latimea pozei

Functia are rolul de a modifica poza intr-o poza grayscale(imagine in tonuri de gri) inlocuind valorile pixelilor (R,G,B) cu (R',G',B') unde R' = G'= B' = 0.299 ∗ R + 0.587 ∗ G + 0.114 ∗ B.

```
int i,j;
for( i = 0 ; i < inaltime_img ; ++i)
    for( j = 0 ; j < latime_img ; ++j)
    {
        unsigned char aux = 0.299*Poza[i*latime_img*3+j*3+2] + 0.587*Poza[i*latime_img*3+j*3+1] + 0.114*Poza[i*latime_img*3+j*3];
        Poza[i*latime_img*3+j*3]   = aux;
        Poza[i*latime_img*3+j*3+1] = aux;
        Poza[i*latime_img*3+j*3+2] = aux;
    }
```

Functia struct fereastra * template_matching(char * nume_imagine,char * nume_cifra,double prag,int *nr_ferestre,int cf)

- nume_imagine reprezinta numele imaginii pe care cautam pattern-urile
- nume_cifra este numele cifrei (sablonul) pe care o aplicam pe imagine
- prag este valoare definita in proiect de 0.5 dupa care separam ferestrele depinzand de corelatie
- nr_ferestre este numarul de ferestre pe care le-am gasit ca respectand conditia de corelatie > prag
- cf reprezinta cifra care o aplicam pe imagine (cifra din sablon)

Functia are rolul de a returna toate ferestrele care respecta conditia de a avea corelatia > prag.

In interiorul functiei avem un vector Poza care tine poza liniarizata dupa ce apelam functia liniarizare_poza si ii facem grayscale-ul imagini apeland functia make_image_grayscale si avem un vector Sablon care tine poza liniarizata dupa ce apelam functia liniarizare_poza si ii facem grayscale-ul imagini apeland functia make_image_grayscale.

Aplicam formulele pentru a calcula corelatia

$$corr(S, f_I) = \frac{1}{n} \sum_{(i,j) \in S} \frac{1}{\sigma_{f_I} \sigma_S} (f_I(i,j) - \bar{f_I}) \ (S(i,j) - \bar{S}), unde$$

- n reprezintă numărul de pixeli în șablonul S (în particular pentru șabloanele utilizate de dimensiuni 11 × 15 pixeli avem n= 11 * 15 = 165);
- indicii i și j reprezintă linia i și coloana j în șablonul S (15 linii și 11 coloane);
- $S(i,j)$ reprezintă valoarea intensității grayscale a pixelului de la linia i și coloana j în șablonul S. Pentru o imagine grayscale, un pixel $P = (p^R, p^G, p^B)$ este reprezentat de un triplet cu toate valorile egale $p^R = p^R = p^B$. În acest caz valoarea intensității grayscale a lui P este p^R;
- $\bar{S}$ reprezintă media valorilor intensităților grayscale a pixelilor în fereastra S (media celor 165 de pixeli din șablonul S);
- σ_S reprezintă deviația standard a valorilor intensităților grayscale a pixelilor în șablonul S: $\sigma_S = \sqrt{\frac{1}{n-1} \sum_{(i,j) \in S} (S(i,j) - \bar{S})^2}$
- $f_I(i,j)$ reprezintă valoarea intensității grayscale a pixelului de la linia i și coloana j în fereastra f_I.
- $\bar{f_I}$ reprezintă media valorilor intensităților grayscale a pixelilor din fereastra f_I (media celor 165 de pixeli din fereastra f_I);
- σ_{f_I} reprezintă deviația standard a valorilor intensităților grayscale a pixelilor în fereastra f_I: $\sigma_{f_I} = \sqrt{\frac{1}{n-1} \sum_{(i,j) \in f_I} (f_I(i,j) - \bar{f_I})^2}$

La sfarsit dezalocam memoria.

Functia int cmp(const void *a,const void *b) unde:

- a reprezinta o fereastra
- b reprezinta o fereastra

Rolul functiei este de a ne ajuta sa sortam eficient ferestrele in ordine descrescatoare a valorilor corelatiei lor.

Functia void sortare_ferestre(struct fereastra *Ferestre,int nr_ferestre) are rolul de a sorta folosind qsort ferestrele din Ferestre acestea fiind in numar de nr_ferestre.

Functia void desenez_contur_fereastra(unsigned char * Poza,struct fereastra Fereastra,unsigned char *RGB,int inaltime_img,int latime_img) unde:

- Vectorul Poza reprezinta poza retinuta sub o forma liniarizata
- Fereastra reprezinta o fereastra al carei contur trebuie sa-l desenam
- Vectorul RGB retine culoarea cu care trebuie sa desenam conturul
- inaltime_img reprezinta inaltimea imaginii
- latime_img reprezinta latimea imaginii

Rolul functiei este de a desena cu o anumita culoare conturul unei ferestre.

Functia void eliminare_non_maxime(struct fereastra *Ferestre, char * ok, int nr_ferestre) unde:

- Vectorul Ferestre reprezinta ferestrele selectate
- Vectorul ok reprezinta stare unei ferestre (0 daca nu se intersecteaza, 1 daca se intersecteaza)
- nr_ferestre reprezinta numarul de ferestre selectate

Rolul functiei este de a elimina ferestrele care au suprapunerea spatiala > 0.2.

Functia void pattern_matching()

In aceasta functie citim numele pozei, sabloanelor(cifrelor) si numele pozei pe care o salvam cu detectiile din fisierul "template_matching_name_file.txt", adun in vectorul Ferestre toate ferestrele cu o corelatie mai mare ca prag dupa care le sortez folosind functia sortare_ferestre si apoi elimin maximele cu ajutorul functiei eliminare_non_maxime.

Citesc poza pe care urmez sa desenez contururile ferestrelor care au ramas. Le desenez pe rand apeland functia desenez_contur_fereastra. La sfarsit afisez poza rezultata si dezaloc memoria.

In main() se apeleaza functia criptare_decriptare si functia pattern_matching.