WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

WYDZIAŁ CYBERNETYKI

Steganografia
Lab. 6

Student
Bartosz Miazga

Prowadzący laboratoria:

Spis treści

Treść zadania

Instrukcje

1. Napisać skrypt w programie Matlab wczytujący obraz o 256 odcieniach szarości.
2. Wykorzystując kodowanie syndromami kodu Hamminga (7,4) i algorytm LSB dokonać ukrycia ciągu tekstowego podanego przez użytkownika w obrazie.
3. Utworzyć skrypt pozwalający na odczytanie wprowadzonych danych.

Zadanie zwrócić w postaci sprawozdania opisującego wykonane zadanie oraz kodów źródłowych (m-pliku). W programie Teams zamieścić tylko pliki nieskompresowane - bez archiwów (ZIP, RAR).

Kod realizujący zadanie

Zadanie zrealizowano w postaci 2 plików: Lab6.m oraz decrypt.m .

Poniżej zaprezentowano zawartość skryptu wraz z komentarzami:

Plik Lab6.m

Lab6.m | decrypt.m

```
image = imread('cat.jpg');
[h , w , l] = size(image);


% zapisanie wiadomości jako ciąg binarny oraz zapamiętanie długości w bitach
wiadomosc = '716';
ascii_value = uint8(wiadomosc);
binary_message = transpose(dec2bin(ascii_value, 8));
binary_message = binary_message(:);
len_binary_message = length(binary_message);
binary_num_message = str2num(binary_message);
%disp(binary_num_message)


start = 1;
rowCounter = 1;
for j = 1 : 1 : w
    % jeśli wiadomości nie uda się zapisać w jednej linijce obrazu to przeskok do kolejnej
    if j == w
        rowCounter = rowCounter + 1;
    end

    % c - słowo kodowe
    bitFromImage = dec2bin(image(j,rowCounter,1),7);
    sevenBits = bitFromImage;
    c = sevenBits;
    c = c.';

    % H - macierz kontroli parzystosci
    H = [1 0 1 0 1 0 1; 0 1 1 0 0 1 1; 0 0 0 1 1 1 1];

    s = H*c; % syndrom

    % zamiana ciągu na ciąg 0 i 1, w zależności od parzystości, parzystosc - 0, nieparzystosc - 1
    zeroOneList = [];
    for i=1 : 1 : length(s)
        if mod(s(i),2) == 0
           zeroOneList(i) = mod(s(i),2);
        end
        if mod(s(i),2) == 1
            zeroOneList(i) = mod(s(i),2);
        end
    end
    s = zeroOneList;

    % jeśli zapisaliśmy całą wiadomość to koniec wykonywania pętli
    if start+2 > len_binary_message
        break;
    end

    % sekret - te bity ukrywamy w słowie kodowym
    sekret = binary_num_message(start:start+2);
    start = start+3;

    roznice = [0 0 0]; % nie ma roznicy
    for i=1 : 1 : length(sekret)
        if sekret(i) ~= s(i)
           roznice(i) = 1;
        end
    end

    % zmiana bitow na przeciwne tam gdzie to konieczne
    bitDoZmiany = [0 0 0 0 0 0 0];
    seven = [1 1 1];
    six = [0 1 1];
    five = [1 0 1];
    four = [0 0 1];
    three = [1 1 0];
    two = [0 1 0];
    one = [1 0 0];

    if roznice == three
       bitDoZmiany = [0 0 1 0 0 0 0];
    end
    if roznice == two
        bitDoZmiany = [0 1 0 0 0 0 0];
    end
    if roznice == one
        bitDoZmiany = [1 0 0 0 0 0 0];
    end
    if roznice == four
```

```
82          bitDoZmiany = [0 0 0 1 0 0 0];
83      end
84      if roznice == five
85          bitDoZmiany = [0 0 0 0 1 0 0];
86      end
87      if roznice == six
88          bitDoZmiany = [0 0 0 0 0 1 0];
89      end
90      if roznice == seven
91          bitDoZmiany = [0 0 0 0 0 0 1];
92      end
93
94      slowo_kodowe = [c(1) c(2) c(3) c(4) c(5) c(6) c(7)];
95
96      % zamiana bitow na przeciwne dla odpowiednich x
97      for i = 1 : 1 : length(slowo_kodowe)
98          if bitDoZmiany(i) == 1
99              if slowo_kodowe(i) == '1'
100                 slowo_kodowe(i) = '0';
101             elseif slowo_kodowe(i) == '0'
102                 slowo_kodowe(i) = '1';
103             end
104         end
105     end
106
107     % slowo kodowe po zmianie
108     c = slowo_kodowe;
109
110     number = bin2dec(string(c));
111     % zapis nowego słowa kodowego w obrazie
112     image(j,rowCounter,1) = number;
113
114 end
115
116 % zapis obrazu
117 imwrite(image,'stegano.png')
118
```

Plik decrypt.m

```
image = imread('stegano.png');
[h , w , l] = size(image);

len_binary_message = 24;
start = 1;
rowCounter = 1;

for i = 1 : 1 : w
    if i == w
        rowCounter = rowCounter + 1;
    end

    % odczytanie z obrazu nowych słów kodowych
    bitFromImage = dec2bin(image(i,rowCounter,1),7);
    sevenBits = bitFromImage;
    c = sevenBits;
    c = c.';

    H = [1 0 1 0 1 0 1; 0 1 1 0 0 1 1; 0 0 0 1 1 1 1]; % macierz kontroli parzystosci

    s = H*c; % syndrom

    % zamiana na ciąg zer i jedynek w zależności od parzystości 1
    zeroOneList = [];
    for i=1 : 1 : length(s)
        if mod(s(i),2) == 0
           zeroOneList(i) = mod(s(i),2);
        end
        if mod(s(i),2) == 1
            zeroOneList(i) = mod(s(i),2);
        end
    end
    s = zeroOneList;

    extracted_bits(start, 1) = s(1);
    extracted_bits(start+1, 1) = s(2);
    extracted_bits(start+2, 1) = s(3);

    if start+3 > len_binary_message
        break;
    end

    start = start + 3;

end


%potęgi liczby 2 do odzyskania znaków ascii z binarki
binValues = [ 128 64 32 16 8 4 2 1 ];

%odkodowanie wiadomości
binMatrix = reshape(extracted_bits, 8,(len_binary_message/8));
textString = char(binValues*binMatrix);
disp(textString)
```

Działanie programu

Pierwszy plik wczytuje obraz, dokonuje ukrycia w nim krótkiego ciągu znaków „716” i zapisuje go jako osobny obraz. Drugi plik ma za zadanie wczytać powstały w wyniku działania skryptu pierwszego obraz i odczytać z niego ukrytą w nim wiadomość.

Poniżej przedstawiono wynik działania programów

```
>> Lab6
>> decrypt
716
fx >>
```

Dodatkowo przedstawione zostały 2 obrazy – po lewej obraz przed ukryciem w nim wiadomości, po prawej obraz po ukryciu w nim wiadomości.


Opis programu

Działanie programu najlepiej opisuje sam kod oraz zawarte w nim komentarze. Starałem się dodać na tyle dużo komentarzy, aby kod był czytelny i zrozumiały. Podczas przygotowywania kodu korzystałem z wykładów. Najtrudniejszym elementem było zaimplementowanie logiki z poniższego zdjęcia:

- Obliczony syndrom *s=010* różni się od wiadomości na wszystkich trzech bitach.
- Zgodnie z układem:

$$\begin{cases} x_1 + x_3 + x_5 + x_7 = s_1 \\ x_2 + x_3 + x_6 + x_7 = s_2 \\ x_4 + x_5 + x_6 + x_7 = s_3 \end{cases}$$

  należy zmienić bit x_7 na przeciwny.
- Nowe słowo kodowe ma postać *1001000*.

W tym celu napisałem poniżej zaprezentowany kod.

```
roznice = [0 0 0]; % nie ma roznicy
for i=1 : 1 : length(sekret)
    if sekret(i) ~= s(i)
       roznice(i) = 1;
    end
end

% zmiana bitow na przeciwne tam gdzie to konieczne
bitDoZmiany = [0 0 0 0 0 0 0];
seven = [1 1 1];
six = [0 1 1];
five = [1 0 1];
four = [0 0 1];
three = [1 1 0];
two = [0 1 0];
one = [1 0 0];

if roznice == three
   bitDoZmiany = [0 0 1 0 0 0 0];
end
if roznice == two
    bitDoZmiany = [0 1 0 0 0 0 0];
end
if roznice == one
    bitDoZmiany = [1 0 0 0 0 0 0];
end
if roznice == four
   bitDoZmiany = [0 0 0 1 0 0 0];
end
if roznice == five
   bitDoZmiany = [0 0 0 0 1 0 0];
end
if roznice == six
   bitDoZmiany = [0 0 0 0 0 1 0];
end
if roznice == seven
   bitDoZmiany = [0 0 0 0 0 0 1];
end

slowo_kodowe = [c(1) c(2) c(3) c(4) c(5) c(6) c(7)];
```

```
% zamiana bitow na przeciwne dla odpowiednich x
for i = 1 : 1 : length(slowo_kodowe)
    if bitDoZmiany(i) == 1
        if slowo_kodowe(i) == '1'
           slowo_kodowe(i) = '0';
        elseif slowo_kodowe(i) == '0'
           slowo_kodowe(i) = '1';
        end
    end
end
```

Najpierw tworzyłem 3 cyfrową listę, która byłą wypełniana zerami i jedynkami w zależności od tego, czy ciąg bitów do ukrycia różnił się względem syndromu. Następnie na podstawie układu rónań utworzyłem listy, które symbolizowały dla których równań (pierwsze, drugie, trzecie) które x są wspólne. Przykładowo x7 występuje we wszystkich równaniach więc seven składa się z trzech

jedynek. Następnie porównywałem wektor różnic z wektorami zmiennych x dla równań i na tej podstawie wyłaniałem odpowiedni bit do zmiany w słowie kodowym. Na sam koniec – funkcja po lewej dokonywała zamiany bitu na przeciwny, w miejscu, w którym powinno się tej zmiany dokonać.