

POLITECHNIKA KRAKOWSKA  
Im. Tadeusza Kościuszki

Wydział Inżynierii Elektrycznej i Komputerowej

Kierunek: Informatyka

**Dokumentacja  
Projektu z przedmiotu:  
Grafika Komputerowa i Multimedia  
Temat:  
Steganografia**

Krzysztof Brożek  
Dawid Budzyński

**1.Wstęp:**

**Steganografia** to nauka o komunikacji w taki sposób, by obecność komunikatu nie mogła zostać wykryta. Wykorzystywana jest do ukrywania informacji m.in w tekstach, obrazach, plikach video, a także dźwiękowych. W odróżnieniu od kryptografii (tam obecność komunikatu nie jest negowana, natomiast jego treść jest niejawna), steganografia próbuje ukryć fakt prowadzenia komunikacji.

Można podzielić ją na trzy rodzaje:  
~ steganografia czysta;  
~ steganografia z kluczem prywatnym (do szyfrowania i odszyfrowania wiadomości wszystkie strony używają tego samego klucza) / tzw. szyfr symetryczny;  
~ steganografia z kluczem publicznym (w tym wypadku do szyfrowania wiadomości używa się klucza publicznego, natomiast do odszyfrowania - klucza prywatnego);

Jedną z metod steganograficznych jest modyfikacja najmniej znaczącego bitu. Ostatni bit zastępowany jest bitem z wiadomości (po uprzedniej modyfikacji przez klucz, na przykład przez użycie bramki logicznej XOR).

XOR (exclusive OR) to alternatywa wykluczająca, której działanie przedstawia tabela:

|  |  |  |
| --- | --- | --- |
| P | Q | P XOR Q |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Tab. 1

Gdzie przy kodowaniu P to bit wiadomości, a Q bit klucza, natomiast przy dekodowaniu P to bit zakodowanej wiadomości w składowych (r/g/b) poszczególnych pikseli (1 bit = 1 składowa).

Przy podwójnym zastosowaniu operacji XOR otrzymujemy z powrotem poprawną wartość bitu wiadomości (oczywiście z tym samym kluczem).

Dzięki temu szyfrowanie i odszyfrowanie to zasadniczo jedna procedura. Jedyna zmiana to uzależnienie P od tego, czy przykładowo szyfrujemy wiadomość tekstową w obrazie (wiadomość XOR - klucz), czy też odszyfrowujemy z powrotem do wiadomości tekstowej (klucz XOR - obraz).

1.a) Przykład szyfrowania wiadomości za pomocą XOR:

Wiadomość: „Ala ma kota”  
01000001 01101100 01100001 00100000 01101101 01100001 00100000 01101011 01101111 01110100 01100001

Klucz: „NIE”

01001110 01001001 01000101

Dla „A” oraz „N”

0 ^ 0 = 0;   
1 ^ 1 = 0;

0 ^ 0 = 0;

0 ^ 0 = 0;

0 ^ 1 = 1;

0 ^ 1 = 1;

0 ^ 1 = 1;

1 ^ 0 = 1;

Wynik:  
00001111 00100101 00100100 01101110 00100100 00100100 01101110 00100010 00101010 00111010 00101000

1.b) Deszyfrowanie:  
Dla pierwszych ośmiu bitów kolejnych składowych (r/g/b) kolejnych pixeli oraz „N”

0 ^ 0 = 0

0 ^ 1 = 1;

0 ^ 0 = 0;

0 ^ 0 = 0;

1 xor 1 = 0;

1 xor 1 = 0;

1 xor 1 = 0;

1 xor 0 = 1;

Wynik: „Ala ma kota”

01000001 01101100 01100001 00100000 01101101 01100001 00100000 01101011 01101111 01110100 01100001

**2. Struktura programu:**  
  
W programie zostało stworzonych 5 klas:  
~ Interface  
 - wybór wersji programu (kodowanie / dekodowanie)

- uzyskanie klucza (hasła)

- uzyskanie nazwy pliku .bmp do steganografii  
- uzyskanie wiadomości do ukrycia

~ Image

- obsługa biblioteki SDL (operacje na plikach bmp)

- sprawdzanie parametrów wczytywanych obrazów bmp

~ Steganography

- Tworzenie vectora pikseli do algorytmu właściwego,

- Algorytmy właściwy (code, decode) oraz pomocnicze dla przejrzystości (changePix, getTextBit),

- Zapis utworzonych plików

~ Conversions

- Konwersje między typami danych (string->bit, bit->string, string->char, bit->int, int->bit, unsigned short->bit, bit-> unsigned short),

~ Files

- Kontrola dodawania rozszerzenia do wpisanej nazwy pliku,

- Wyszukiwanie plików w katalogu programu z zadanym rozszerzeniem,

- Zwracanie adresu katalogu z programem,

Input programu posiada następujące wymogi:

- plik bmp musi posiadać przynajmniej 15 pikseli

- wymiary pliku bmp nie mogą przekroczyć 1920x1080 pikseli

- rozmiar wiadomości nie może przekraczać 777595 znaków

- wiadomość nie może zawierać polskich znaków

**Steganography::code**Po wybraniu opcji pierwszej w menu, program pobiera potrzebne dane i przesyła je do Steganography::code(). Tam określa rozmiar bitowy hasła oraz wiadomości, a następnie tworzy wektor pikseli potrzebnych, by w pełni zakodować wiadomość. Następnie ustawia początkowe wartości zmiennych pomocniczych i przechodzi do wykonywania algorytmu właściwego Steganography::changePix().   
Wykonuje on dwie pętle while(1), gdzie pierwsza ma za zadanie zakodować rozmiar wiadomości na pierwszych 11 pikselach, druga natomiast koduje wiadomość. Rozmiar wiadomości w typie long zajmuje 32 bity, podczas gdy 11 pikseli to 33 składowe r/g/b, wobec czego pomijana jest wartość składowa red piksela o indeksie 0. Dzięki temu można przewidzieć, że kodowanie rozmiaru zakończy się, gdy wykona się changePix dla składowej blue w przejściu p = 11 (atrybuty klasy Steganography const int messageSizeBmp = 11). Przy kodowaniu rozmiaru nie używa się hasła, zamiast tego do metody changePix jako trzeci argument odsyłana jest wartość 1.

while (1){

changePix(pixelsArray[p].g, bitwiseMessageLengthB[w++], 1);

changePix(pixelsArray[p++].b, bitwiseMessageLengthB[w++], 1);

if (p == 11) break;

changePix(pixelsArray[p].r, bitwiseMessageLengthB[w++], 1);

}

Po zakodowaniu rozmiaru, program przechodzi do kodowania wiadomości właściwej. Tym razem trzecim argumentem jest bit hasła, dlatego po każdym wykonaniu metody changePix należy sprawdzić, czy nie był on ostatnim bitem hasła. Jeśli tak, hasło jest powielane. Należy również sprawdzać za każdym razem, czy bit wiadomości nie był jej ostatnim. W takim wypadku program wychodzi z pętli while i przechodzi do zapisu zmienionego obrazu.

while (1){

changePix(pixelsArray[p].r, messageB[w++], passwordB[h++]);

if (w == bitwiseMessageLength) break;

if (h == bitwisePasswordLength) h = 0;

changePix(pixelsArray[p].g, messageB[w++], passwordB[h++]);

if (w == bitwiseMessageLength) break;

if (h == bitwisePasswordLength) h = 0;

changePix(pixelsArray[p].b, messageB[w++], passwordB[h++]);

if (w == bitwiseMessageLength) break;

if (h == bitwisePasswordLength) h = 0;

p++;  
}

Ciało Steganography::changePix - głównego algorytmu programu, odpowiedzialnego za wykonanie operacji XOR:

void Steganography::changePix(Uint8 & pix, bool number, bool number2) {

usint.clear(); //reset wartosci vectora

unsigned short a = (unsigned short)pix; //rgb -> unsigned short int

usint = converObj.ushortToBits(a); //unsigned short int -> wektor z bitami

usint[0] = (number ^ number2); //xor

pix = (Uint8)converObj.bitsToUShort(usint); //wektor -> unsigned short int -> rgb

}

**Steganography::decode:**

Po wybraniu opcji drugiej w menu, program pobiera potrzebne dane i przesyła je do Steganography::decode(). Tam określa rozmiar bitowy hasła, a następnie tworzy wektor 11 pikseli potrzebnych, by odszyfrować rozmiar wiadomości. Odczytuje pierwsze 32 bity zapisane na 11 pikselach za pomocą Steganography::getTextBit() i konwertuje wartość z wektora bitowego na int long.

while (1){

TheLength.push\_back(getTextBit(first11Pix[p].g, 1));

TheLength.push\_back(getTextBit(first11Pix[p++].b, 1));

if (p == 11) break;

TheLength.push\_back(getTextBit(first11Pix[p].r, 1));

}

long length = converObj.bitToInt(TheLength);

Następnie tworzy vektor pikseli potrzebny do odszyfrowania wiadomości i przechodzi do wykonywania algorytmu właściwego. Zarówno w Steganography::code(), jak i tutaj należy jednocześnie sprawdzać, czy już wszystkie bity wiadomości zostały odczytane, oraz czy należy powielić hasło.

while (1){

TheMessage.push\_back(getTextBit(pixelsArray[p].g, passwordB[h++]));

if (++w == length) break;

if (h == bitwisePasswordLength) h = 0;

TheMessage.push\_back(getTextBit(pixelsArray[p++].b, passwordB[h++]));

if (++w == length) break;

if (h == bitwisePasswordLength) h = 0;

TheMessage.push\_back(getTextBit(pixelsArray[p].r, passwordB[h++]));

if (++w == length) break;

if (h == bitwisePasswordLength) h = 0;

}

Ciało Steganography::getTextBit - głównego algorytmu programu, odpowiedzialnego za wykonanie operacji XOR:

bool Steganography::getTextBit(Uint8 pix, bool number2){

usint.clear(); //reset wartosci wektora

unsigned short a = (unsigned short)pix; //rgb -> unsigned short int

usint = converObj.ushortToBits(a); //unsigned short int -> wektor z bitami

bool textbit = (usint[0] ^ number2); //xor

return textbit; //zwrocenie pierwotnego bitu przez push\_back()

}

Po wykonaniu algorytmu głównego i uzyskaniu vektor<bool> z wiadomością, program konwertuje kontener do typu string i zapisuje pierwotną wiadomość do nowo utworzonego pliku tekstowego.

**Bibliografia:**https://en.wikipedia.org/wiki/Steganography  
https://prezi.com/o\_-rg-kn2qgf/steganografia/  
<http://stackoverflow.com/>

<http://en.cppreference.com>

https://wiki.libsdl.org/