Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Лабораторная работа № 4

**«Разработка программы шифрования/дешифрирования текста с**

**использованием методов стеганографии.»**

## Отчёт по дисциплине

**«Методы защиты КС»**

Пенза 2020

**Выполнили с-ты гр. 17ВВ2:**

Живаев В.  
Безжонов Р.

**Приняли:**

к.т.н. Дубравин А.В.  
Кармышева Н.С.

**Цель работы.**

Научиться разрабатывать программы шифрования/дешифрования текста с использованием перестановочного шифра.

**Задание.**

Разработать программу, выполняющую шифрование ирасшифровывание произвольного текстового файла с использованиемперестановочного шифра используя в качестве ключа последовательность,соответствующую номеру варианта (табл. 1). Выполнить проверку путем двоичногосравнения исходного файла и фала, полученного после расшифровывания.

**Листинг.**

#include "stdafx.h"

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <fstream>

#include <Windows.h>

#include <locale.h>

using namespace std;

#define MAX\_SIZE\_MSG 64\*1024

typedef unsigned char byte;

char crcString[2];

char msgLen[2];

wchar\_t\* imgFileName = L"5.bmp";

wchar\_t\* msgFileName = L"toCrypt.txt";

int absoluteCurPosition = 0;

int curBitPos = 0;

int curImagePos = 0;

int imgI = 0;

int imgJ = 0;

BITMAPINFOHEADER infoHeader;

BITMAPFILEHEADER fileHeader;

HANDLE bmp, msgFile;

char buffer[MAX\_SIZE\_MSG];

DWORD dwReaded;

int imageSize;

struct Pixel

{

byte red;

byte green;

byte blue;

} CUR\_PIXEL;

int imageCapacity(BITMAPINFOHEADER\* fileInfo) {

return fileInfo->biHeight\*fileInfo->biWidth \* 2 / 8;

}

int getBit(char block, int numBit)

{

return ((block & (1 << numBit)) != 0);

}

unsigned short crc16(unsigned char\* data\_p, int length) {

unsigned char x;

unsigned short crc = 0xFFFF;

while (length--) {

x = crc >> 8 ^ \*data\_p++;

x ^= x >> 4;

crc = (crc << 8) ^ ((unsigned short)(x << 12)) ^ ((unsigned short)(x << 5)) ^ ((unsigned short)x);

}

return crc;

}

char\* calcCrc(unsigned char\* data\_p, int length) {

USHORT block = crc16(data\_p, length);

char\* crc;

crc = (char\*)malloc(sizeof(char) \* 3);

crc[0] = (block >> 8) & 255;

crc[1] = (block) & 255;

crc[2] = '\0';

return crc;

}

char\* msgLenToChar(USHORT len) {

char\* msglen = (char\*)malloc(sizeof(char) \* 3);

msglen[0] = (len >> 8) & 255;

msglen[1] = (len) & 255;

msglen[2] = '\0';

return msglen;

}

USHORT getUSHORT(char buffer[2]) {

unsigned short block = 0;

block |= buffer[0] << 8 & 0xff00;

block |= buffer[1] & 0x00ff;

return block;

}

int writeCurrentPixel() {

bmp = CreateFile(imgFileName, GENERIC\_WRITE, FILE\_SHARE\_READ, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

if (bmp == INVALID\_HANDLE\_VALUE) {

printf("cannot read or open image file");

return 0;

}

SetFilePointer(bmp, absoluteCurPosition - 3, 0, FILE\_BEGIN);

if (!WriteFile(bmp, &CUR\_PIXEL, sizeof(CUR\_PIXEL), &dwReaded, 0)) {

CloseHandle(bmp);

return 0;

}

CloseHandle(bmp);

return -1;

}

int readCurrentPixel() {

bmp = CreateFile(imgFileName, GENERIC\_READ, FILE\_SHARE\_READ, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

if (bmp == INVALID\_HANDLE\_VALUE) {

printf("\nCannot read or open image file");

CloseHandle(bmp);

return 0;

}

SetFilePointer(bmp, absoluteCurPosition, 0, FILE\_BEGIN);

if (!ReadFile(bmp, &CUR\_PIXEL, sizeof(Pixel), &dwReaded, 0)) {

printf("\nerror reading pixel");

CloseHandle(bmp);

return 0;

}

absoluteCurPosition += 3;

if (++curImagePos > infoHeader.biWidth) {

curImagePos = 0;

//SetFilePointer(bmp, infoHeader.biWidth % 4, 0, FILE\_CURRENT);

//absoluteCurPosition += infoHeader.biWidth % 4;

}

CloseHandle(bmp);

return 1;

}

int cryptAndRewriteChar(char ch) {

for (int i = 0; i < 8; i += 2) {

if (!readCurrentPixel()) {

printf("\nerror reading pixel");

return 0;

}

char value = getBit(ch, i);

CUR\_PIXEL.red = (CUR\_PIXEL.red & 0xfe) | value;

value = getBit(ch, i + 1);

CUR\_PIXEL.green = (CUR\_PIXEL.green & 0xfe) | value;

if (!writeCurrentPixel()) {

printf("\nerror writing pixel");

return 0;

}

}

return 1;

}

void printMenu() {

system("cls");

printf("\t\t\tSTEGANOGRAPHY");

printf("\n\t\t\1)Зашифровать собщение в файл");

printf("\n\t\t\2)Прочитать сообщение из файла");

printf("\n\t\t\3)Выход");

}

int cryptMessage(char \*message) {

int len = strlen(message);

if (strlen(message) > imageCapacity(&infoHeader)) {

printf("\nРазмер сообщения не соответствует размеру изображения.");

return 0;

}

if (len == 0 && message[1] != 0) {

for (int i = 0; i < 2; i++) {

if (!cryptAndRewriteChar(message[i])) {

printf("\n error crypt char");

return 0;

}

}

}

else {

for (int i = 0; i < len; i++) {

if (!cryptAndRewriteChar(message[i])) {

printf("\n error crypt char");

return 0;

}

}

}

return 1;

}

char readChar() {

char chToBuild = 0;

for (int i = 0; i < 8; )

{

if (!readCurrentPixel()) {

printf("Cannot read pixel while fetching char");

return 0;

}

chToBuild |= (CUR\_PIXEL.red & 0x01) << i++;

chToBuild |= (CUR\_PIXEL.green & 0x01) << i++;

}

return chToBuild;

}

char\* readData(int len) {

char \*data = (char\*)malloc(sizeof(char) \* len + 1);

memset(data, 0, len + 1);

for (int i = 0; i < len; i++) {

char readed = readChar();

data[i] = readed;

}

return data;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

#pragma region init

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

bmp = CreateFile(imgFileName, GENERIC\_READ, FILE\_SHARE\_READ, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

if (bmp == INVALID\_HANDLE\_VALUE) {

printf("cannot read or open image file");

return -1;

}

msgFile = CreateFile(msgFileName, GENERIC\_READ, FILE\_SHARE\_READ, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

if (bmp == INVALID\_HANDLE\_VALUE) {

printf("cannot open msg file");

return -1;

}

if (!ReadFile(msgFile, &buffer, MAX\_SIZE\_MSG, &dwReaded, 0)) {

printf("cannot Read msg");

return -1;

}

if (!ReadFile(bmp, &fileHeader, sizeof(BITMAPFILEHEADER), &dwReaded, 0)) {

printf("cannot read or open image file");

return -1;

}

if (!ReadFile(bmp, &infoHeader, sizeof(BITMAPINFOHEADER), &dwReaded, 0)) {

printf("cannot read or open image file");

return -1;

}

if (strlen(buffer) + 4 > imageCapacity(&infoHeader)) {

printf("\nРазмер сообщения не соответствует размеру изображения.");

return 0;

}

SetFilePointer(bmp, fileHeader.bfOffBits, 0, FILE\_BEGIN);

absoluteCurPosition = fileHeader.bfOffBits;

CloseHandle(bmp);

#pragma endregion

while (true) {

printMenu();

char key = \_getch();

switch (key) {

case '1': {

#pragma region crypt

bool hasErrors = false;

if (!cryptMessage(calcCrc((unsigned char\*)buffer, strlen(buffer)))) {

printf("\nОшибка. Не удалось записать контрольную сумму.");

hasErrors = true;

};

if (!cryptMessage(msgLenToChar(strlen(buffer))) && !hasErrors) {

printf("\nОшибка. Не удалось записать длину сообщения.");

};

if (!cryptMessage(buffer) && !hasErrors) {

printf("\nОшибка. Не удалось записать сообщение.");

};

absoluteCurPosition = fileHeader.bfOffBits;

if (!hasErrors) {

printf("\nУспех.");

}

\_getch();

break;

#pragma endregion

}

case '2': {

#pragma region decrypt

bool hasErrors = false;

char\* crc = readData(2);

char\* msglen = readData(2);

USHORT len = getUSHORT(msglen);

if (len + 4 > imageCapacity(&infoHeader)) {

printf("Файл не содержит сообщения. Длина %d явно превышает максимальную вместимость.");

hasErrors = true;

}

char\* message = readData(len);

if (strcmp(calcCrc((unsigned char\*)message, len), crc) && !hasErrors) {

printf("Несовпадение контрольной суммы. Файл не содержит сообщения.");

hasErrors = true;

}

if (!hasErrors) {

printf("\n\t\tReaded message: %s", message);

printf("\n\t\tУспех.");

}

\_getch();

#pragma endregion

break;

}

case '3':

goto end;

}

}

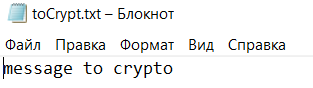
end:

return 0;

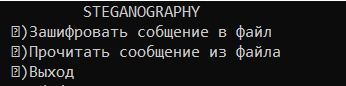
}

**Результат работы.**

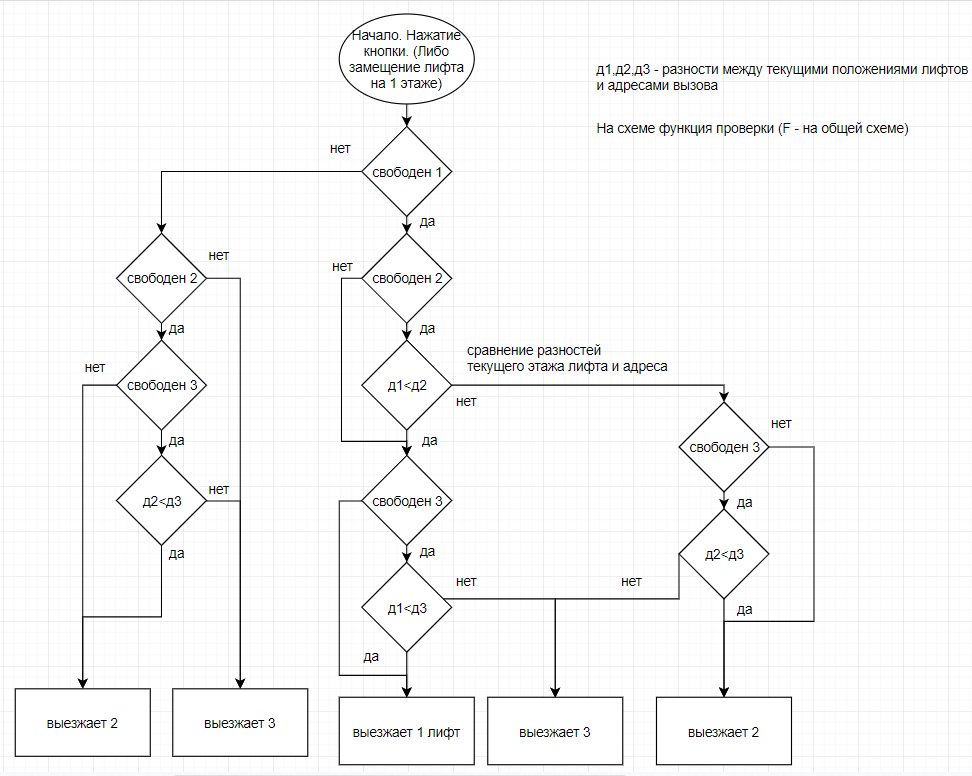
Результат работы программы показан на рисунках 1-5.



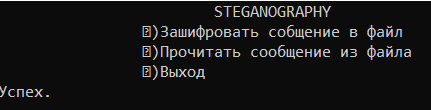
**Рисунок 1 - Исходный файл.**



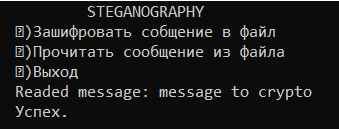
**Рисунок 2 - Меню программы.**



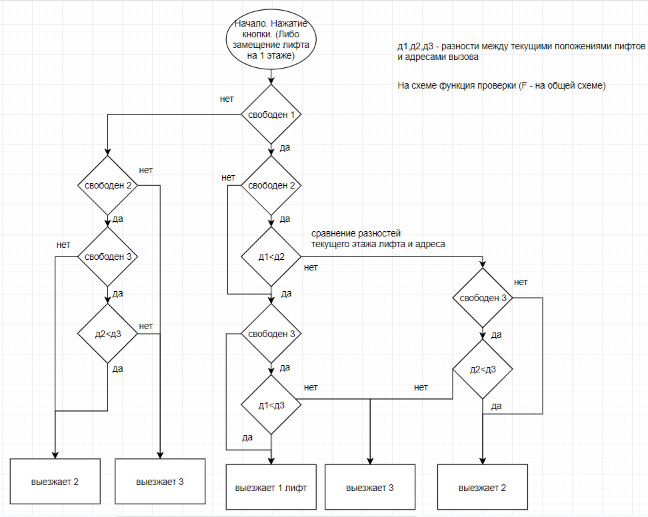
**Рисунок 3 – Изображение до шифрования.**



**Рисунок 4 - Результат шифрование. Вывод**.



**Рисунок 5 - результат чтения сообщения из файла**



**Рисунок 6 - Изображение после дешифрования.**

Исходный текст файла совпал с результатом извлечения из изображения.

**Вывод.**

Научились разрабатывать программы шифрования/дешифрования текста с использованием методов стеганографии и алгоритма LSB.