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.**

// 4.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <cstdio>

#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\* outputFile = L"5.bmp";

wchar\_t\* inputFile = L"4.exe";

LONGLONG fileLength;

int absoluteCurPosition = 0;

int curBitPos = 0;

int curImagePos = 0;

int imgI = 0;

int imgJ = 0;

bool encrypt = false;

bool decrypt = false;

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() {

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

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

printf("Ошибка записи шифрованного пикселя в файл %d", GetLastError());

return 0;

}

return -1;

}

int readCurrentPixel() {

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;

}

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 messageLength) {

if (messageLength > imageCapacity(&infoHeader)) {

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

return 0;

}

if (messageLength == 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 < messageLength; 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;

}

bool isKey(wchar\_t\* arg) {

bool flag = false;

flag = lstrcmpW(arg, L"-e") ? false : true;

flag = lstrcmpW(arg, L"--encrypt") ? false : true;

flag = lstrcmpW(arg, L"-d") ? false : true;

flag = lstrcmpW(arg, L"--decrypt") ? false : true;

flag = lstrcmpW(arg, L"-h") ? false : true;

flag = lstrcmpW(arg, L"--help") ? false : true;

return flag;

}

bool isPresent(wchar\_t\* arg) {

if (arg != NULL) {

return true; //проверка наличия аргумента

}

printf("Не передан аргумент. Справка -h|--help");

exit(666);

}

bool isKeyValid(wchar\_t\* arg[], int paramsCount) {

isPresent(arg[0]);

for (int i = 1; i <= paramsCount; i++) {

if (isPresent(arg[i])) { //аргумент команды должен быть представлен и не должен являться ключем

isKey(arg[i]);

}

}

return true;

}

void help() {

printf("Для управления используются следующие ключи:\n");

printf("\t\t -h --help вызов помощи\n");

printf("\t\t -e --encrypt [msgFile] [bmpFile] сокрытие входного файла в изображении\n");

printf("\t\t -d --decrypt [bmpFiile] проверка наличия и извлечение сокрытого сообщения\n\t\tрезультат в файле \"decrypted\"");

exit(1);

}

void argsParser(int argc, wchar\_t\* argv[]) {

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

if (lstrcmpW(argv[i], L"-h") == 0 || lstrcmpW(argv[i], L"--help") == 0) {

help();

}

if (lstrcmpW(argv[i], L"-e") == 0 || lstrcmpW(argv[i], L"--encrypt") == 0) {

if (isKeyValid(&argv[i], 2)) {

if (decrypt) {

printf("Можно вызвать только одну процедуру -d/-e");

exit(666);

}

encrypt = true;

inputFile = argv[i + 1];

outputFile = argv[i + 2];

}

}

if (lstrcmpW(argv[i], L"-d") == 0 || lstrcmpW(argv[i], L"--decrypt") == 0) {

if (isKeyValid(&argv[i], 1)) {

if (encrypt) {

printf("Можно вызвать только одну процедуру -d/-e");

exit(666);

}

decrypt = true;

inputFile = argv[i + 1];

}

}

}

}

void printMessage(char\* message, int len) {

HANDLE decrypted = CreateFile(L"decrypted", GENERIC\_ALL, FILE\_SHARE\_READ, 0, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, 0);

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

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

exit(666);

}

if (!WriteFile(decrypted, message, len, &dwReaded, 0)) {

printf("Ошибка записи расшифрованного файла %d", GetLastError());

CloseHandle(decrypted);

exit(666);

}

CloseHandle(decrypted);

}

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

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

#pragma region init

argsParser(argc, argv);

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

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

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

return -1;

}

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

if (msgFile == INVALID\_HANDLE\_VALUE && !decrypt) {

printf("cannot open msg file");

return -1;

}

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

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 (!decrypt) {

DWORD dwSizeH = 0, dwSizeL = 0;

dwSizeL = GetFileSize(msgFile, &dwSizeH);

fileLength = ((LONGLONG)dwSizeH \* ((LONGLONG)MAXDWORD + 1)) + dwSizeL;

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

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

return 0;

}

}

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

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

absoluteCurPosition = fileHeader.bfOffBits;

#pragma endregion

#pragma region crypt

if (encrypt) {

bool hasErrors = false;

if (!cryptMessage(calcCrc((unsigned char\*)buffer, fileLength),2)) {

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

hasErrors = true;

};

if (!cryptMessage(msgLenToChar(fileLength),2) && !hasErrors) {

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

};

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

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

};

absoluteCurPosition = fileHeader.bfOffBits;

if (!hasErrors) {

printf("Успех.");

}

\_getch();

}

#pragma endregion

#pragma region decrypt

if (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.");

free(message);

hasErrors = true;

}

if (!hasErrors) {

printMessage(message, len);

free(message);

printf("Успех.");

}

\_getch();

}

#pragma endregion

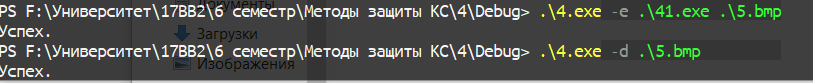
CloseHandle(bmp);

return 0;

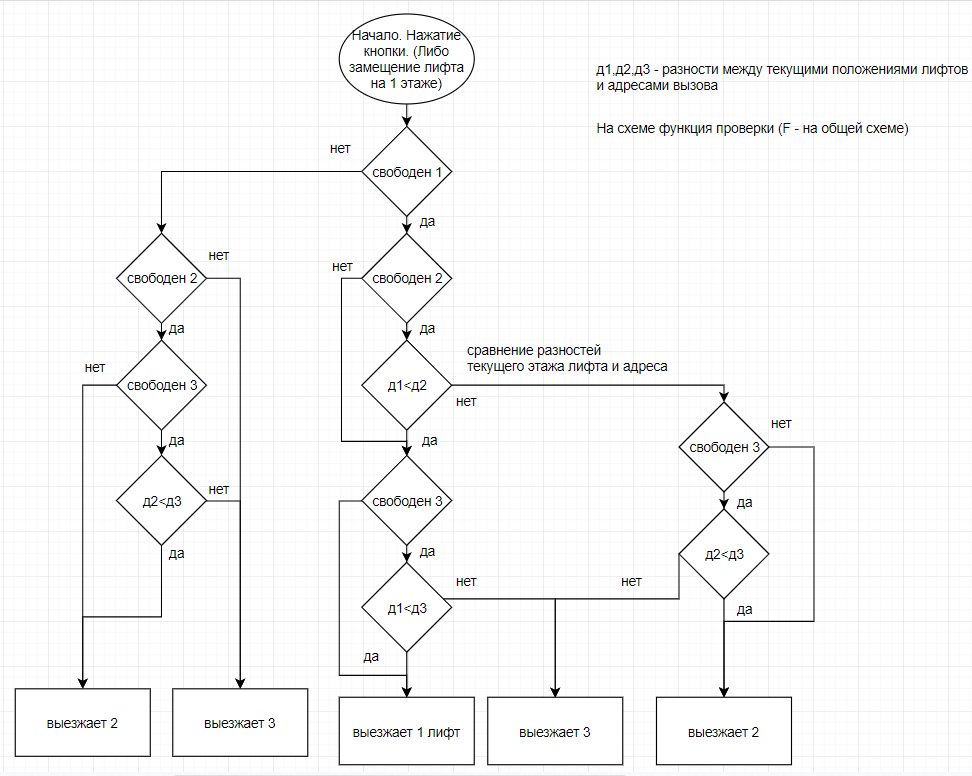
}

**Результат работы.**

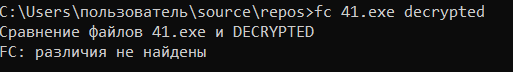
Результат работы программы показан на рисунках 1-5.

****

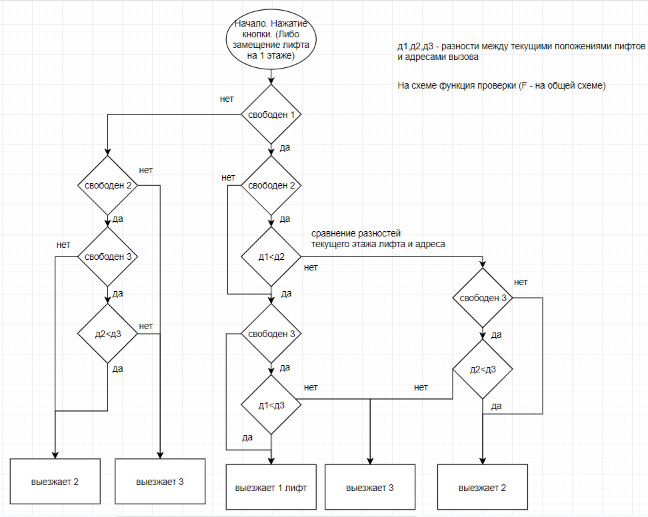
**Рисунок 1 – Процессы сокрытия и извлечения.**



**Рисунок 2 – Изображение до шифрования.**

****

**Рисунок 3 - результат сравнения сообщения и извлеченных данных**



**Рисунок 4 - 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.