МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт КНТ

Факультет ФИСП

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №2

Тема: «Базовые алгоритмы шифрования. Использование симметричного алгоритма «Сеть Фейстеля» для шифрования файлов на диске»

Курс: Безопасность программ и данных

Выполнил

ст. гр. ПИ-18б

Моргунов А.Г.

Проверил

Ногтев Е.А.

Донецк – 2021

ЗАДАНИЕ

Задание к лабораторной работе

В соответствии с вариантом задания написать программу, реализующую:

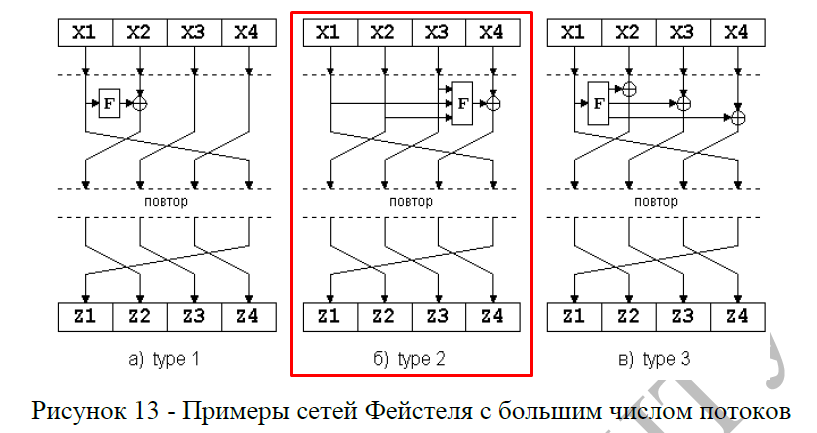
1) Шифрование данных с помощью одного из базовых алгоритмов шифрования (см.варианты заданий). При этом произвести некоторую модификацию выбранного базового алгоритма.

2) Зашифрованные с помощью базового алгоритма данные затем зашифровать с помощью алгоритма «Сеть Фейстеля». В качестве схемы «Сети Фейстеля» взять схему в соответствии с вариантом заданий.

3) В качестве функции F в сети Фейстеля (см. схему сети Фейстеля по варианту заданий) взять XOR с ключом (размер ключа не менее размера блока данных в "Сети Фейстеля").

4) Длина блока для сети Фейстеля - 128 бит, длина потока – 32 бит, кол-во раундов не менее 8.

5) Программа должна выполнять чтение файла, просмотр, его шифрование, расшифрование, просмотр зашифрованного и расшифрованного документа.



Варианты заданий:

№ по журналу Базовый алгоритм Схема сети Фейстеля

- 5 Шифр Цезаря; рис. 13(б)

ОПИСАНИЕ ПРОГРАММЫ

Базовый алгоритм – Шифр Цезаря. Заключается в том, что имеется алфавит и ключ шифра(N). Каждый символ заменяется на другой символ в алфавите, который стоит на N пунктов дальше. Например в русском алфавите шифр Цезаря с ключем 1 будет заменять А на Б, Б на В, В на Г и т.п.

В данной программе алфавит состоит из русских букв, английских букв, цифр и пробела. Ключ задается пользователем.

Схема сети Фейстеля указана в задании.

Алгоритм программы следующий.

Пользователь вводит ключи для шифра Цезаря и шифра Фейстеля. Количество раундов сети Фейстеля равно 8. Размер блока данных 128 бит (16 байт), блок данных делится на 4 части по 4 байта. Символы в кодировке UTF-8 занимают по 2 байта, т.е. в каждой части помещаются по 2 символа.

После нажатия на кнопку «Зашифровать» информация считывается из исходного файла и отображается в первом текстовом поле программы. После чего шифруется шифром Цезаря и шифром Фейстеля. Зашифрованное сообщение выводится во второй текстовый блок, а также записывается в файл “encryptedFile.txt”. После этого зашифрованное сообщение расшифровывается в обратном порядке и выводится пользователю в третий текстовый блок. Также расшифрованное сообщение сохраняется в файле “decryptedFile.txt”.

СКРИНШОТЫ

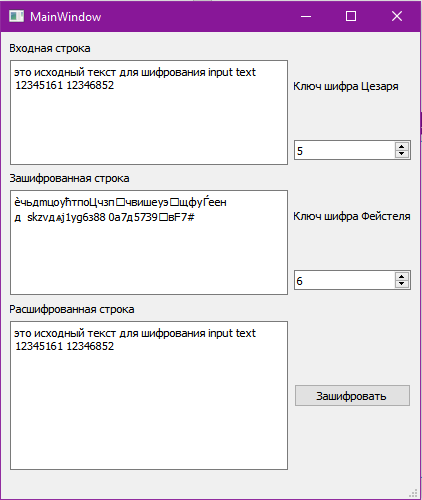


Рисунок 1 – Окно программы

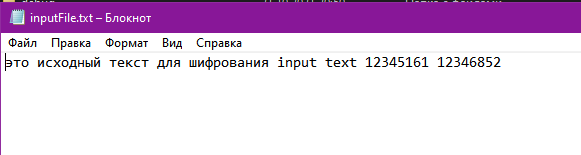


Рисунок 2 – Содержание начального файла

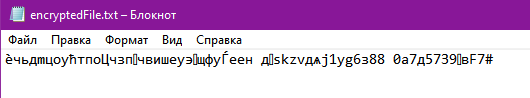


Рисунок 3 – Содержание зашифрованного файла

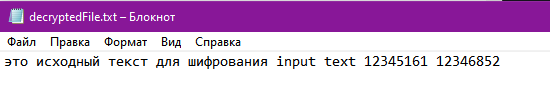


Рисунок 4 – Содержание расшифрованного файла

ПРОГРАММНЫЙ КОД

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QString>

#include <QDebug>

#include <QTextStream>

#include <iostream>

#include <QFile>

#include <QTextCodec>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

SetupMap(*alphabet*);

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

void MainWindow::**on\_pushButton\_clicked**()

{

QString string = ReadFromFile(

"E:/Projects/C++/Qt/BPD\_lab\_2/inputFile.txt");

*//* *SetTextFromFile();*

ui->input->setPlainText(string);

*//* *GetKey();*

int chaesarKey = ui->chaesar\_key->value();

int feistelKey = ui->feistel\_key->value();

*//* *GetRounds();*

*const* int rounds = 8;

QString encrypted = Encrypt(string, chaesarKey, feistelKey, rounds);

ui->encrypted->setPlainText(encrypted);

WriteToFile(encrypted, "encryptedFile.txt");

QString decrypted = Decrypt(encrypted, chaesarKey, feistelKey, rounds);

ui->decrypted->setPlainText(decrypted);

WriteToFile(decrypted, "decryptedFile.txt");

}

void MainWindow::**SetupMap**(std::map <int, QChar> &map) {

int i = 0;

map[i++] = ' ';

*for* (wchar\_t letter = L'а'; letter <= L'я'; letter++, i++){

*if* (letter == L'ж'){

map[i++] = L'ё';

}

map[i] = letter;

}

*for* (wchar\_t letter = 'a'; letter <= 'z'; letter++, i++){

map[i] = letter;

}

*for* (wchar\_t letter = '0'; letter <= '9'; letter++, i++){

map[i] = letter;

}

*//* *qDebug()* *<<* *map;*

}

QString MainWindow::**ChaesarCipherEncrypt**(QString string, int key) {

std::map <int, QChar>::iterator it;

QString encryptedLine;

*for* (QChar ch : string) {

it = std::find\_if(alphabet.begin(), alphabet.end(),

[ch](std::pair <int, QChar> it){

*return* it.second == ch;

});

*if* (it == alphabet.end()) {

std::terminate();

}

*//* *qDebug()* *<<* *QString(it->first);*

int currentIndex = it->first;

int newIndex = (currentIndex + key) % alphabet.size();

int offset = newIndex - currentIndex;

*//* *qDebug()* *<<* *currentIndex* *<<* *"* *>>* *"* *<<* *newIndex;*

std::advance(*it*, offset);

QChar encrypterChar = it->second;

encryptedLine.append(encrypterChar);

}

*return* encryptedLine;

}

QString MainWindow::**ChaesarCipherDecrypt**(QString encryptedLine, int key) {

std::map <int, QChar>::iterator it;

QString decryptedLine;

*for* (QChar ch : encryptedLine) {

it = std::find\_if(alphabet.begin(), alphabet.end(),

[ch](std::pair <int, QChar> it){

*return* it.second == ch;

});

*if* (it == alphabet.end()) {

std::terminate();

}

*//* *qDebug()* *<<* *QString(it->first);*

int currentIndex = it->first;

int distanceFromZero = currentIndex - key % alphabet.size();

int newIndex = (distanceFromZero + alphabet.size()) % alphabet.size();

int offset = newIndex - currentIndex;

*//* *qDebug()* *<<* *currentIndex* *<<* *"* *>>* *"* *<<* *newIndex;*

std::advance(*it*, offset);

QChar decrypterChar = it->second;

decryptedLine.append(decrypterChar);

}

*return* decryptedLine;

}

QString MainWindow::**FeistelCipherEncrypt**(QString string, int key, int rounds) {

std::vector<QChar> chars = QStringToQCharVector(string);

std::vector<ushort> codes1 = QCharVectorToUshortVector(chars);

std::vector<std::uint32\_t> codes = UshortToUint32Vector(codes1);

*for* (int i = 0; i < codes.size(); i+=4){

*for* (int j = 0; j < rounds; ++j){

ushort xor\_for\_123 = (codes[i] ^ codes[i+1]) ^ codes[i+2];

ushort xor\_with\_key = xor\_for\_123 ^ key;

++key;

codes[i+3] ^= xor\_with\_key;

std::swap(*codes[i]*, *codes[i+*1*]*);

std::swap(*codes[i+*1*]*, *codes[i+*2*]*);

std::swap(*codes[i+*2*]*, *codes[i+*3*]*);

*//* *qDebug()* *<<* *"i* *=* *"* *<<* *i* *<<* *"* *:* *"* *<<* *codes;*

}

*//* *qDebug()* *<<* *"\n";*

}

codes1 = Uint32VectorToUshortVector(codes);

QString encryptedLine;

*for* (ushort ch : codes1) {

encryptedLine.append(QChar(ch));

}

*return* encryptedLine;

}

QString MainWindow::**FeistelCipherDecrypt**(QString string, int key, int rounds) {

std::vector<QChar> chars = QStringToQCharVector(string);

std::vector<ushort> codes1 = QCharVectorToUshortVector(chars);

std::vector<std::uint32\_t> codes = UshortToUint32Vector(codes1);

int lastKey = key + rounds \* codes.size() / 4 - 1;

*for* (int i = codes.size()-1; i > 0; i-=4){

*for* (int j = 0; j < rounds; ++j) {

std::swap(*codes[i-*1*]*, *codes[i]*);

std::swap(*codes[i-*2*]*, *codes[i-*1*]*);

std::swap(*codes[i-*3*]*, *codes[i-*2*]*);

ushort xor\_for\_123 = codes[i-3] ^ codes[i-2] ^ codes[i-1];

ushort xor\_with\_key = xor\_for\_123 ^ lastKey;

--lastKey;

codes[i] ^= xor\_with\_key;

*//* *qDebug()* *<<* *"i* *=* *"* *<<* *i-3* *<<* *"* *:* *"* *<<* *codes;*

}

*//* *qDebug()* *<<* *"\n";*

}

codes1 = Uint32VectorToUshortVector(codes);

QString decryptedLine = UShortVectorToString(codes1);

*return* decryptedLine;

}

std::vector<QChar> MainWindow::**QStringToQCharVector**(QString string){

std::vector<QChar> chars;

QChar \*data = string.data();

*while* (!data -> isNull()) {

chars.push\_back(\*data);

++data;

}

*return* chars;

}

std::vector<ushort> MainWindow::**QCharVectorToUshortVector**(

std::vector<QChar> chars) {

std::vector<ushort> codes;

*for* (QChar ch : chars){

codes.push\_back(ch.unicode());

}

FillVectorWithZero(*&codes*);

*return* codes;

}

*//* *Дозаполнить* *вектор* *нулями* *до* *128* *битной* *границы*

void MainWindow::**FillVectorWithZero**(std::vector<ushort> \*vector) {

int blockSize = 16;

*while* (vector->size() \* *sizeof*(ushort) % blockSize != 0){

vector->push\_back(QChar(0).unicode());

}

}

QString MainWindow::**UShortVectorToString**(std::vector<ushort> vector) {

QString string;

*for* (ushort ch : vector) {

*if*(ch != 0)

string.append(QChar(ch));

}

*return* string;

}

QString MainWindow::**ReadFromFile**(QString filePath) {

QFile file(filePath);

*if*(!file.*open*(QIODevice::*ReadOnly*)){

qDebug() << "open file error";

std::terminate();

}

QTextStream input(*&file*);

input.setCodec("UTF-8");

QString string = input.readAll();

file.*close*();

*return* string;

}

QString MainWindow::**Encrypt**(QString string, int chaesarKey,

int feistelKey, int rounds) {

QString chaesarEncryptedString = ChaesarCipherEncrypt(string, chaesarKey);

qDebug() << chaesarEncryptedString;

QString feistelEncryptedString = FeistelCipherEncrypt(

chaesarEncryptedString, feistelKey, rounds);

qDebug() << feistelEncryptedString;

*return* feistelEncryptedString;

}

QString MainWindow::**Decrypt**(QString string, int chaesarKey,

int feistelKey, int rounds) {

QString feistelDecryptedString = FeistelCipherDecrypt(

string, feistelKey, rounds);

qDebug() << feistelDecryptedString;

QString startString = ChaesarCipherDecrypt(feistelDecryptedString,

chaesarKey);

qDebug() << startString;

*return* startString;

}

std::vector<std::uint32\_t> MainWindow::**UshortToUint32Vector**(std::vector<ushort> vector) {

std::vector<std::uint32\_t> int32vector;

*for* (int i = 0; i < vector.size(); i += 2) {

int32vector.push\_back(

(*static\_cast*<std::uint32\_t>(vector[i+1]) << 16) | vector[i]);

}

*return* int32vector;

}

std::vector<ushort> MainWindow::**Uint32VectorToUshortVector**(std::vector<std::uint32\_t> vector) {

std::vector<ushort> ushortVector;

*for* (int i = 0; i < vector.size(); ++i) {

ushortVector.push\_back(*static\_cast*<ushort>(vector[i]));

ushortVector.push\_back(*static\_cast*<ushort>(vector[i] >> 16));

}

*return* ushortVector;

}

void MainWindow::**WriteToFile**(QString text, QString filename){

QFile file(filename);

*if*(!file.*open*(QIODevice::*WriteOnly*)) {

*return* ;

}

QTextStream out(*&file*);

out.setCodec("UTF-8");

out << text;

out.flush();

file.*close*();

}

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

*return* a.exec();

}

}

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <map>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private* slots:

void **on\_pushButton\_clicked**();

*private*:

Ui::MainWindow \*ui;

std::map <int, QChar> alphabet;

void **SetupMap**(std::map <int, QChar> &map);

QString **ChaesarCipherEncrypt**(QString string, int key);

QString **ChaesarCipherDecrypt**(QString string, int key);

QString **FeistelCipherEncrypt**(QString string, int key, int rounds);

QString **FeistelCipherDecrypt**(QString string, int key, int rounds);

std::vector<QChar> **QStringToQCharVector**(QString string);

std::vector<ushort> **QCharVectorToUshortVector**(std::vector<QChar> chars);

void **FillVectorWithZero**(std::vector<ushort> \*vector);

QString **UShortVectorToString**(std::vector<ushort> vector);

QString **ReadFromFile**(QString fileName);

QString **Encrypt**(QString string, int chaesarKey, int feistelKey, int rounds);

QString **Decrypt**(QString string, int chaesarKey, int feistelKey, int rounds);

std::vector<std::uint32\_t> **UshortToUint32Vector**(std::vector<ushort> vector);

std::vector<ushort> **Uint32VectorToUshortVector**(std::vector<std::uint32\_t> vector);

void **WriteToFile**(QString text, QString filename);

};

#endif *//* *MAINWINDOW\_H*