Лабораторная работа

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ

Цель работы: разработать программу криптографической защиты символьной последовательности методом одноразового блокнота.

Методические указания

1. Основные понятия криптографии

Разработкой методов преобразования (шифрования) информации с целью ее защиты от незаконных пользователей занимается криптография. Такие методы и способы преобразования информации называются шифрами.

Шифрование (зашифрование) – процесс применения шифра к защищаемой информации, т. е. преобразование защищаемой информации (открытого текста) в шифрованное сообщение (шифртекст, криптограмму) с помощью определенных правил, содержащихся в шифре.

Составными элементами шифра являются:

– алфавиты для записи исходных сообщений (защищаемой информации, открытого текста) и шифрованных сообщений (шифртекстов, шифрограмм, криптограмм);

– алгоритмы криптографического преобразования (зашифрования и дешифрования);

– множество ключей.

Азбука или алфавит (греч. ἀλφάβητος) – форма письменности, основанная на стандартном наборе знаков, один или набор которых соответствуют фонемам1 языка. В общем случае алфавит для записи исходных сообщений и алфавит для записи шифрованных сообщений могут отличаться. Например, исходные сообщения записываются с помощью букв, а шифрограммы с помощью цифр или графических обозначений.

Алгоритмы шифрования и зашифрования, как правило, отличаются друг от друга, но могут и совпадать. В частности, в некоторых комбинированных блочных шифрах (DES, ГОСТ 28147-89) алгоритмы совпадают, но отличаются порядком использования ключа (ключевых элементов). Алгоритм шифрования

может включать в себя предварительное кодирование, а алгоритм расшифрования – обратное перекодирование. Например, в аддитивных шифрах перед шифрованием буквы (символы) исходного сообщения заменяются на числа, а результат расшифрования в виде чисел преобразуется в буквы (символы). Аналогичная ситуация имеет место и в некоторых шифрах замены (шифр Хилла), комбинированных блочных шифрах, асимметричных шифрах.

Дешифрование – процесс, обратный шифрованию, т. е. преобразование шифрованного сообщения в защищаемую информацию с помощью определенных правил, содержащихся в шифре.

Под ключом в криптографии понимают сменный элемент шифра, который применяется для шифрования конкретного сообщения.

2. Метод шифрования Вернана (одноразовый блокнот).

По методу, применяемому для кодирования информации, шифр Вернана относят к аддитивным шифрам. В них используется сложение по модулю (mod) исходного сообщения с гаммой, представленных в числовом виде. Напомним, что результатом сложения двух целых чисел по модулю является остаток от деления (например, 5+10 mod 4 = 15 mod 4 = 3).

В литературе шифры этого класса часто называют потоковыми, хотя к потоковым относятся и другие разновидности шифров. Стойкость закрытия этими шифрами определяется, главным образом, качеством гаммы, которое зависит от длины периода и случайности распределения по периоду [8]. При этом символы в пределах периода гаммы являются ключом шифра.

Длиною периода гаммы называется минимальное количество символов, после которого последовательность цифр в гамме начинает повторяться. Случайность распределения символов по периоду означает отсутствие закономерностей между появлением различных символов в пределах периода.

По длине периода различаются гаммы с конечным и бесконечным периодом. Если длина периода гаммы превышает длину шифруемого текста, гамма является истинно случайной и не используется для шифрования других

сообщений, то такое преобразование является абсолютно стойким (совершенный шифр).

Сложение по модулю N. В 1888 г. француз маркиз де Виари в одной из своих научных статей, посвященных криптографии, доказал, что при замене букв исходного сообщения и ключа на числа справедливы формулы

$$C_i = (P_i + K_i) \bmod N, \qquad (1)$$

$$P_i = (C_i + N - K_i) \bmod N, \qquad (2)$$

где P_i, C_i – i-ый символ открытого и шифрованного сообщения;

N – количество символов в алфавите;

$K_{i–i\text{-ый}}$ символ гаммы (ключа). Если длина гаммы меньше, чем длина сообщения, то она используется повторно.

Данные формулы позволяют выполнить зашифрование / расшифрование по Виженеру при замене букв алфавита числами согласно следующей таблице (применительно к русскому алфавиту):

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Рис. 1 Таблица кодирования символов

Например, для шифрования используется русский алфавит (N = 33), открытое сообщение – «КАФЕДРА», гамма – «ДЕКАНАТ». При замене символов на числа буква А будет представлена как 0, Б – 1, …, Я – 32. Результат шифрования показан в следующей таблице.

Таблица 1

Пример аддитивного шифрования по модулю N = 33

Символ								
открытого сообщения, P_i	А	Б	Р	А	М	О	В
	0	1	17	0	13	15	2
гаммы, K_i	Ж	У	Р	И	Х	И	Н
	7	20	17	9	22	9	14
шифрограммы, C_i	Ж	Ф	Б	И	В	Ч	П
	7	21	1	9	2	24	16

Сложение по модулю 2. Значительный успех в криптографии связан с именем американца Гильберто Вернама [15]. В 1917 г. он, будучи сотрудником

телеграфной компании AT&T, совместно с Мейджором Джозефом Моборном предложил идею автоматического шифрования телеграфных сообщений. Речь шла о своеобразном наложении гаммы на знаки алфавита, представленные в соответствии с телетайпным кодом Бодо пятизначными «импульсными комбинациями». Например, буква А представлялась комбинацией («– – – + +»), а комбинация («+ + – + +») представляла символ перехода от букв к цифрам. На бумажной ленте, используемой при работе телетайпа, знаку «+» отвечало наличие отверстия, а знаку «–» - его отсутствие. При считывании с ленты металлические щупы проходили через отверстия, замыкали электрическую цепь и, тем самым, посылали в линию импульс тока.

Рис. 2. Шифровальная машина Siemens M-190 [www.cryptomuseum.com]

Вернам предложил электромеханически покоординатно складывать «импульсы» знаков открытого текста с «импульсами» гаммы, предварительно нанесенными на ленту. Сложение проводилось «по модулю 2». Имеется в виду, что если «+» отождествить с 1, а «–» с 0, то сложение определяется двоичной арифметикой:

⊕	0	1
0	0	1
1	1	0

При данном способе шифрования символы текста и гаммы представляются в двоичных кодах, а затем каждая пара двоичных разрядов складывается по модулю 2 (⊕, для булевых величин аналог этой операции – XOR, «Исключающее ИЛИ»). Процедуры шифрования и дешифрования выполняются по следующим формулам

$$C_i = P_i \oplus K_i, \qquad (3)$$

$$P_i = C_i \oplus K_i. \qquad (4)$$

Вернам сконструировал и устройство для такого сложения. Замечательно то, что процесс шифрования оказывался полностью автоматизированным, в предложенной схеме исключался шифровальщик. Кроме того, оказывались слитыми воедино процессы зашифрования / расшифрования и передачи по каналу связи.

В 1918 г. два комплекта соответствующей аппаратуры были изготовлены и испытаны. Испытания дали положительные результаты. Единственное неудовлетворение специалистов - криптографов было связано с гаммой. Дело в том, что первоначально гамма была нанесена на ленту, склеенную в кольцо. Несмотря на то, что знаки гаммы на ленте выбирались случайно, при зашифровании длинных сообщений гамма регулярно повторялась. Этот недостаток так же отчетливо осознавался, как и для шифра Виженера. Уже тогда хорошо понимали, что повторное использование гаммы недопустимо даже в пределах одного сообщения.

Рис. 3 Шифровальный блокнот (СССР, ГДР, 1960-е гг.)

Попытки удлинить гамму приводили к неудобствам в работе с длинным кольцом. Тогда был предложен вариант с двумя лентами, одна из которых шифровала другую, в результате чего получалась гамма, имеющая длину периода, равную произведению длин исходных периодов.

Шифры гаммирования стали использоваться немцами в своих дипломатических представительствах в начале 20-х гг., англичанами и американцами – во время Второй мировой войны. Разведчики-нелегалы ряда государств использовали шифрблокноты. Шифр Вернама (сложение по модулю 2) применялся на правительственной «горячей линии» между Вашингтоном и Москвой, где ключевые материалы представляли собой перфорированные бумажные ленты, производившиеся в двух экземплярах.

Следует отметить, что классический одноразовый шифровальный блокнот - большой неповторяющийся случайный набор символов ключа, написанный на листах бумаги, склеенных в блокнот. Шифровальщик при личной встрече снабжался блокнотом, каждая страница которого содержала ключ. Такой же блокнот имелся и у принимающей стороны. Использованные страницы после однократного использования уничтожались.

Перед иллюстрацией использования шифра приведем таблицу кодов символов Windows 1251 и их двоичное представление.

Таблица 2.

Коды символов Windows 1251 и их двоичное представление

Буква	Dec-код	Bin-код	Буква	Dec-код	Bin-код	Буква	Dec-код	Bin-код
А	192	1100 0000	Л	203	1100 1011	Ц	214	1101 0110
Б	193	1100 0001	М	204	1100 1100	Ч	215	1101 0111
В	194	1100 0010	Н	205	1100 1101	Ш	216	1101 1000
Г	195	1100 0011	О	206	1100 1110	Щ	217	1101 1001
Д	196	1100 0100	П	207	1100 1111	Ъ	218	1101 1010
Е	197	1100 0101	Р	208	1101 0000	Ы	219	1101 1011
Ж	198	1100 0110	С	209	1101 0001	Ь	220	1101 1100
З	199	1100 0111	Т	210	1101 0010	Э	221	1101 1101
И	200	1100 1000	У	211	1101 0011	Ю	222	1101 1110
Й	201	1100 1001	Ф	212	1101 0100	Я	223	1101 1111
К	202	1100 1010	Х	213	1101 0101			

Примечание. Dec-код – десятичный код символа, Bin-код – двоичный код символа.

Пример шифрования сообщения «ВОВА» с помощью гаммы «ЮЛЯ» показан в таблице 3.

Таблица 3.

Пример аддитивного шифрования по модулю 2

Открытое сообщение, P_i	Буква	В	О	В	А
	Dec-код	194	206	194	192
	Bin-код	1100 0010	1100 1110	1100 0010	1100 0000
Гамма, K_i	Буква	Ю	Л	Я	Ю
	Dec-код	222	203	223	222
	Bin-код	1101 1110	1100 1011	1101 1111	1101 1110
Шифрограмма, C_i	Dec-код	28	5	29	30
	Bin-код	0001 1100	0000 0101	0001 1101	0001 1110

Задание по лабораторной работе.

1. Разработать программу шифратора/дешифратора методом одноразового блокнот на любом языке программирования, например C++, C#, Python.
2. Программа должна обладать интуитивно-понятным графическим интерфейсом.
3. Пользователь должен ввести с клавиатуры текстовые данные или загрузить текстовый файл.
4. Загруженный текст должен отображаться в окне программы.
5. Ключ для шифрации/дешифрации может быть записан заранее или сгенерирован в процессе работы программы.
6. Результат шифрования требуется вывести рядом с нешифрованным текстом для подтверждения его шифрования.
7. Пользователь должен иметь возможность сохранить полученный при шифровании/дешифровании результат в текстовый файл.
8. Подключить разработанные программы к событиям соответствующих кнопок.
9. Проверить правильность разработанных программ с помощью тестов. Результат прохождения тестов представить на скриншотах

Содержание отчета.

1. Краткие теоретические сведения о выбранном методе шифрования.

2. Приведен укрупненный алгоритм процесса шифрации/дешифрации.

3. Описание пользовательского интерфейса.

4. Представлены описания библиотек и функций, на основе которых разработана программа.

5. Программный код закомментирован приблизительно на 20-25%.

6. Представлены скриншоты работы программы.

7. Присутствуют выводы о проделанной работе и список используемых источников.

8. При защите работы потребуется продемонстрировать ее работу преподавателю и сдать программный код в виде файла.

В приложениях А и Б представлены варианты отчетов по данной лабораторной работе. Программы разработаны с использованием библиотек и функций языка программирования Python и C++.

Пример выполнения работы № 1 на языке программирования Python

1. Цель работы

Разработать программы шифрования и дешифрования текста на основе метода «Блокнот».

2. Описание выполненных действий

2.1. Разработка графического интерфейсаю

В ходе работы был разработан графический интерфейс для удобства общения пользователя с программой (рисунок 1).

На форме расположены такие элементы, как поле для ввода названия файла со словарем, поле для ввода исходного текста, поле для вывода расшифрованного текста, кнопка «Кодировать», кнопка «Декодировать», кнопка «Очистить» и «Сохранить».

В поле «Файл со словарем» необходимо ввести имя и расширение файла, по которому будет производиться кодирование или декодирование текста (файл должен находиться в той же папке, что и программа).

В поле «Исходный текст» вводится текст, который необходимо закодировать или декодировать.

В поле «Результат» выводится результат кодирования или декодирования.

При нажатии на кнопку «Кодировать» производится кодирование введенного текста в поле «Исходный текст», используя указанный словарь в поле «Файл со словарем». Результат работы отобразится в поле «Результат».

По нажатию на кнопку «Декодировать» производится аналогичная процедура, только вместо кодирования происходит декодирование.

По нажатию на кнопку «Очистить» удаляется информация из полей для ввода «Исходный текст» и «Результат».

По нажатию на кнопку «Сохранить» информация из поля «Результат» сохраняется в файл results.txt.

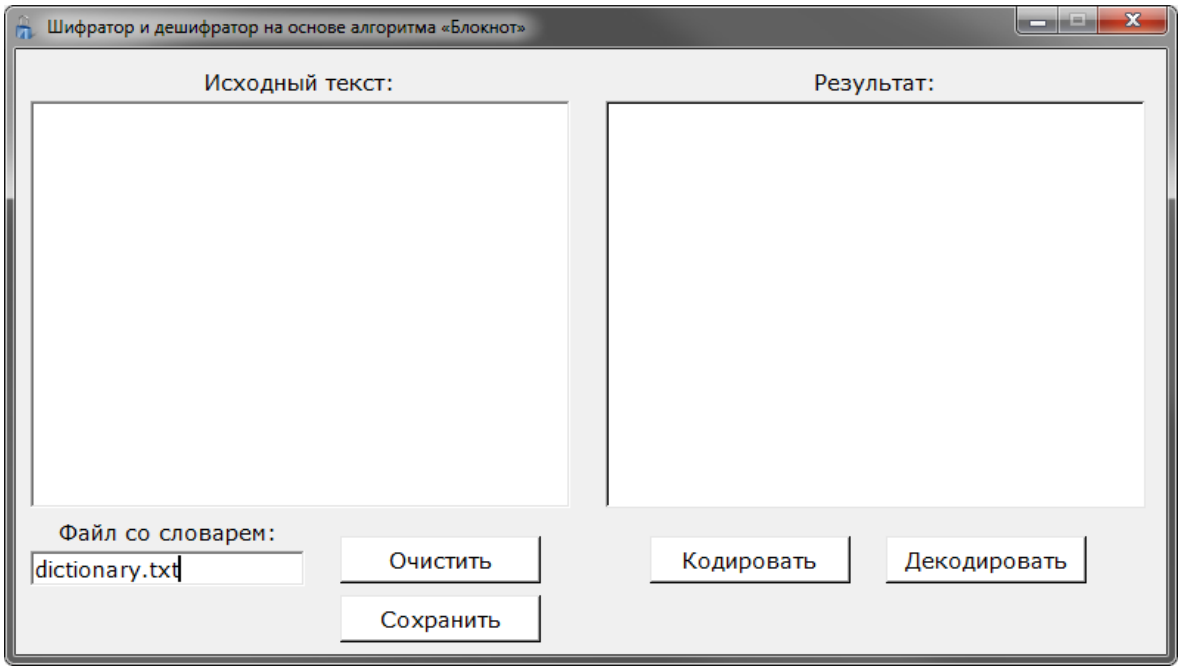

Рисунок 1 – Скриншот графического интерфейса

2.2. Описание библиотек и методов программирования

Программа состоит из класса и нескольких методов внутри него. Первый метод – def initUI(self). Метод отвечает за формирования самой формы графического интерфейса и правильного отображения всех элементов.

Второй метод – def create_pad(self). Метод позволяет пользователю самому задать текст, который будет считаться готовым блокнотом по которому будет происходить шифрование текста.

Третий метод – def id_generator(self, chars = string.ascii_uppercase + string.digits + string.ascii_lowercase + string.punctuation + ' '). Метод генерирует блокнот. Пользователь нажимает соответствующую кнопку на форме и происходит генерация случайных последовательность из строчных и прописных букв, цифр, знаков препинания и пробельных символов.

Четвертый метод – def encode_text(self). Метод отвечает за шифрование текста и вывод на экран соответствующего исходному закодированного текста.

Пятый метод – def decode_text(self). Метод отвечает за правильность декодирования текста и вывод его на экран.

Шестой метод – def write_text(self). Метод отвечает за запись закодированного текста в файл. После успешной записи пользователю выводится информационное сообщение. Метод использует декоратор @pyqtSlot. Это необходимо для явного задания метода Python, как слота. А также такая реализация позволяет уменьшить объем занимаемой памяти и немного сократить время выполнения.

Седьмой и восьмой методы – read_text(self) и read_encode_text(self). Методы отвечают за чтение исходного/кодированного текста и записи его в поле для последующего кодирования/декодирования. Также используют декоратор @pyqtSlot.

Подробный текст программы вынесен в приложение 1.

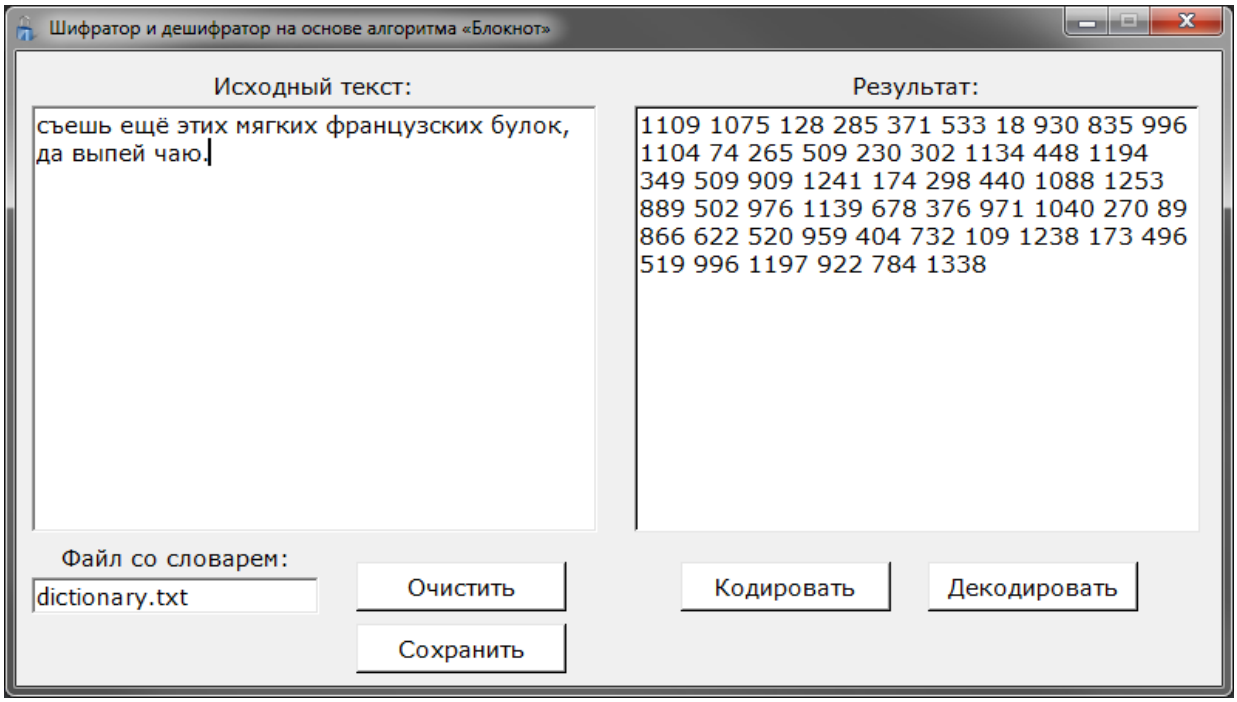

Рисунок 2 – Пример кодирования текста

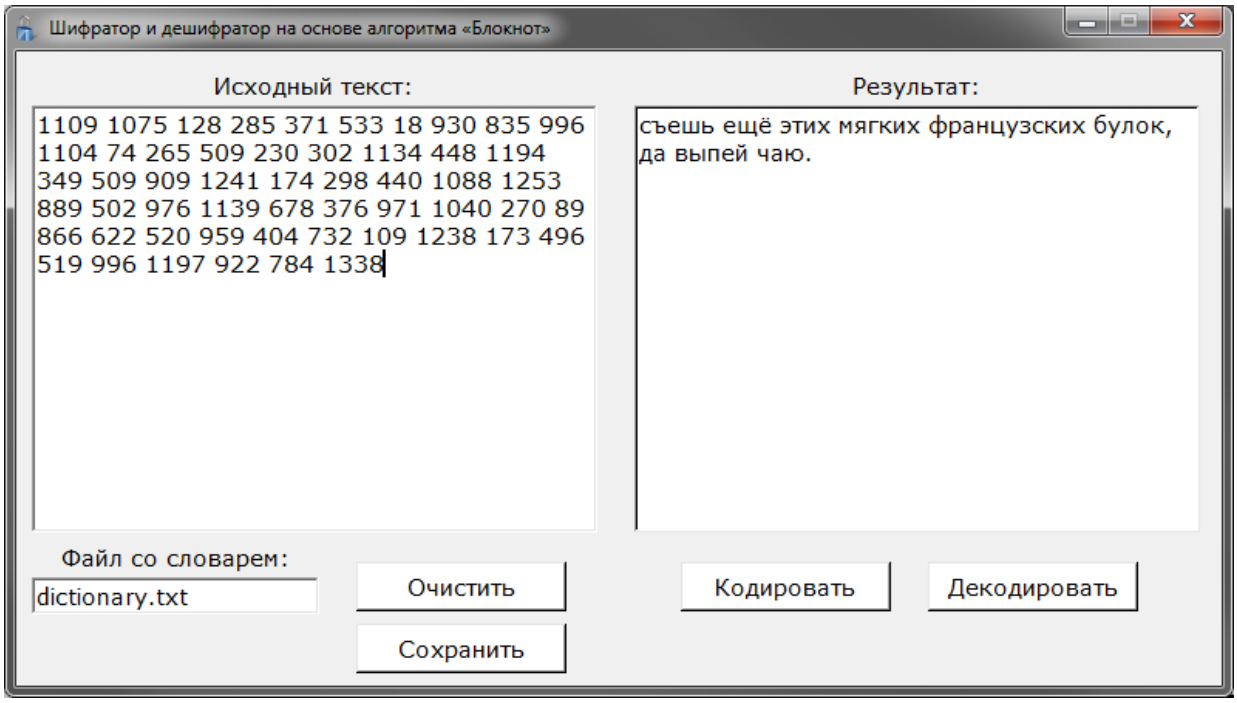

Рисунок 3 – Пример декодирования текста

Если в словаре отсутствуют символы, которые необходимо закодировать – будет выведено сообщение об ошибке.

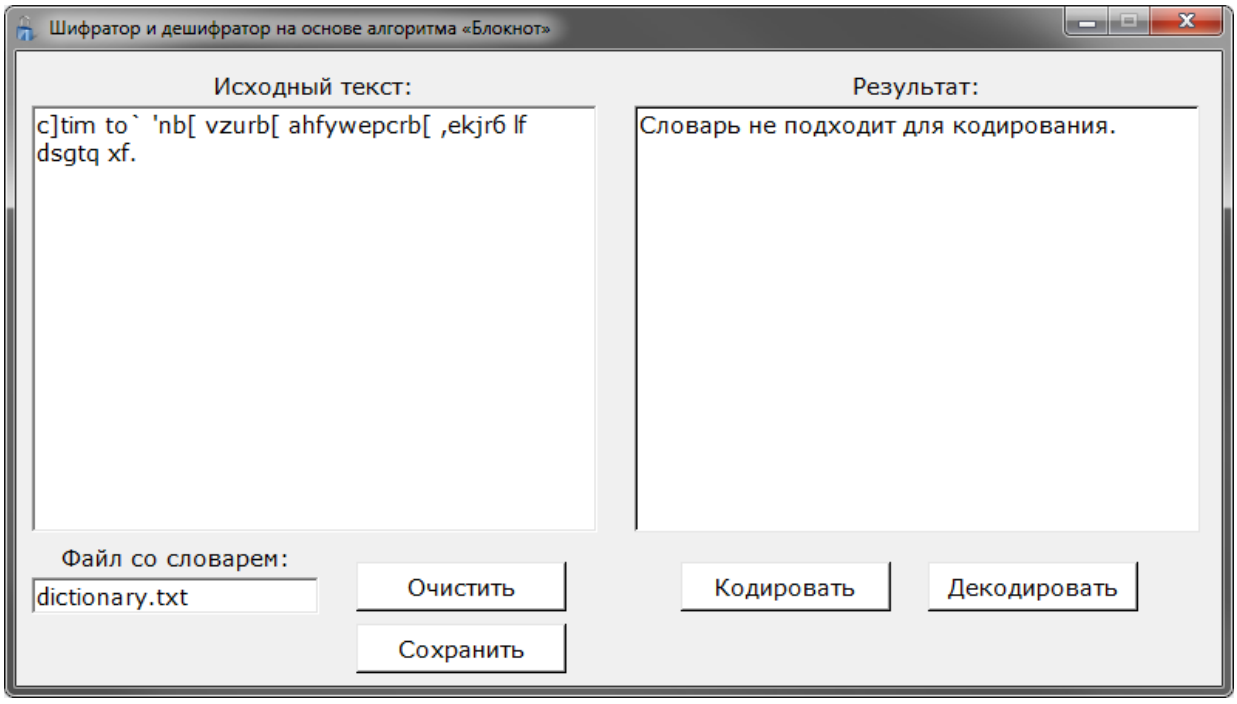

Рисунок 4 – Пример ошибки при кодировании

3. Выводы

В данной работе была разработана программа шифрования и дешифрования текста на основе метода «Блокнот» и разработан графический интерфейс пользователя с поддержкой загрузки и сохранения результатов в файл. Тестирование данной программы показало, что программа работает корректно.

Основным недостатком такого метода является необходимость случайной, а не псевдослучайной последовательности символов для создания блокнота. Кроме того, шифрование для большей надежности не должно быть последовательным (со сдвигом). Также, если объем блокнота недостаточно большой относительно шифруемого текста, с увеличением объема шифруемой информации уменьшается криптостойкость шифра.

Листинг программы

```
# -*- coding: utf-8 -*-
from tkinter import *
from tkinter.filedialog import *
import random

def get_coding(letter, max_rnd, lst):
    rnd_index = random.randint(0, max_rnd)
    rez = -1
    for i in range(rnd_index, len(lst)):
        if lst[i] == letter:
            rez = i
            break
    return rez


def get_lst():
    txt = txt2cod.get('1.0', 'end')
    cod_txt = []
    for i in txt:
        cod_txt.append(str(i))
    k = libEntry.get()
    file = open(libEntry.get(), 'r', encoding="utf8")
    line = file.read()
    lst = []
    for i in line:
        lst.append(str(i))
    return lst, cod_txt

def coding_click(event):
    lst = get_lst()
    max_rnd = round(len(lst[0]) - 0.3*len(lst[0]))
    coding = []
    for i in range(len(lst[1])):
        txt2 = get_coding(lst[1][i], max_rnd, lst[0])
        coding.append(txt2)
    coding = coding[0:len(coding) - 1]
    flag = 0
    for i in range(len(coding)):
        if coding[i] == -1:
            flag = 1
    if flag == 1:
        output.delete('1.0', 'end')
        output.insert("0.0", 'Словарь не подходит для кодирования.')
    else:
        output.delete('1.0', 'end')
        output.insert("0.0", coding)

def get_decoding(num):
    lst = get_lst()
    rez = lst[0][num]
    return rez

def decoding_click(event):
    decod_txt = txt2cod.get('1.0', 'end')
    decod_txt_str = []
    for i in range(len(decod_txt)):
        decod_txt_str.append(str(decod_txt[i]))
    decod_txt_str = decod_txt_str[0:len(decod_txt_str)-1]
    decod_txt_str.append(str(' '))
    decod_txt_int = []
    count = ''
    for i in range(len(decod_txt_str)):
```

```
            if decod_txt_str[i] != ' ':
                count = count + decod_txt_str[i]
            else:
                count_int = int(count)
                decod_txt_int.append(count_int)
                count = ''
                count_int = 0
        print(decod_txt_int)
        lst = get_lst()
        decoding = []
        for i in range(len(decod_txt_int)):
            txt2 = get_decoding(decod_txt_int[i])
            decoding.append(txt2)
        decoding = ''.join(decoding)
        output.delete('1.0', 'end')
        output.insert("0.0", decoding)

    def clear(event):
        output.delete('1.0', 'end')
        txt2cod.delete('1.0', 'end')

    def save(event):
        file = open('results.txt', 'w')
        file.write(output.get('0.0', 'end'))
        file.close()

    window = Tk()
    window.title("Шифратор и дешифратор на основе алгоритма «Блокнот»")
    window.iconbitmap(u'main.ico')
    window.resizable(width=False, height=False)
    window.configure(bg='white')

    frame1 = Frame(window, width=780, height=410)
    frame1.grid(row=0, column=0)

    label1 = Label(frame1, text='Исходный текст:', width=40, font="Verdana
11")
    label1.place(x=10, y=10)
    txt2cod = Text(frame1, width=36, height=15, bd=2, font="Verdana 11",
wrap=WORD)
    txt2cod.config(state=NORMAL)
    txt2cod.place(x=10, y=35)

    label2 = Label(frame1, text='Результат:', width=40, font="Verdana 11")
    label2.place(x=400, y=10)
    output = Text(frame1, bg="white", font="Verdana 11", width=36,
height=15, bd=2, wrap=WORD)
    output.config(state=NORMAL)
    output.place(x=400, y=35)

    label3 = Label(frame1, text='Файл со словарем:', width=20,
font="Verdana 11")
    label3.place(x=10, y=315)
    libEntry = Entry(frame1, width=18, bd=2, font="Verdana 11")
    libEntry.insert(END, 'dictionary.txt')
    libEntry.place(x=10, y=340)

    btnCod = Button(frame1, width=14, bg='white', font="Verdana 11")
    btnCod.place(x=430, y=330)
    btnCod['text'] = 'Кодировать'
    btnCod.bind('<Button-1>', coding_click)

    btnDec = Button(frame1, width=14, bg='white', font="Verdana 11")
    btnDec.place(x=590, y=330)
    btnDec['text'] = 'Декодировать'
    btnDec.bind('<Button-1>', decoding_click)
```

```
btnClear = Button(frame1, width=14, bg='white', font="Verdana 11")
btnClear.place(x=220, y=330)
btnClear['text'] = 'Очистить'
btnClear.bind('<Button-1>', clear)

btnSave = Button(frame1, width=14, bg='white', font="Verdana 11")
btnSave.place(x=220, y=370)
btnSave['text'] = 'Сохранить'
btnSave.bind('<Button-1>', save)
window.mainloop()
```

Пример выполнения работы № 2
на языке программирования C++

Цель: разработать программы шифрования и дешифрования текста на основе метода «Блокнот».

Выполнение работы:

На рисунке 1 представлена форма программы. Здесь шесть кнопок и три поля:

- кнопка выбора файла блокнота-ключа
- кнопка шифрования
- кнопка дешифрования
- кнопка очистки полей
- кнопка сохранения образованного шифр-текста
- кнопка сохранения расшифрованного текста
- поле ввода исходного текста
- поле вывода шифр-текста
- поле вывода расшифрованного текста

Рисунок 1 – Главная форма

До тех пор, пока не выбран ключ-файл блокнота, кнопки шифрования и дешифрования неактивны, о чем свидетельствует их тусклый цвет (рисунок 2).

На нажатие эти кнопки не реагируют. Но после выбора ключ-файла они становятся обычными активными.

Рисунок 2 – Кнопки шифрования и дешифрования неактивны

В ключ-файле находится заранее занесенный туда текст. Это может быть как набор случайных символов, так и осмысленный текст. Например, внесем туда отрывок из первой главы книги «Война и мир», в котором много символов как латиницы, так и кириллицы (рисунок 3).

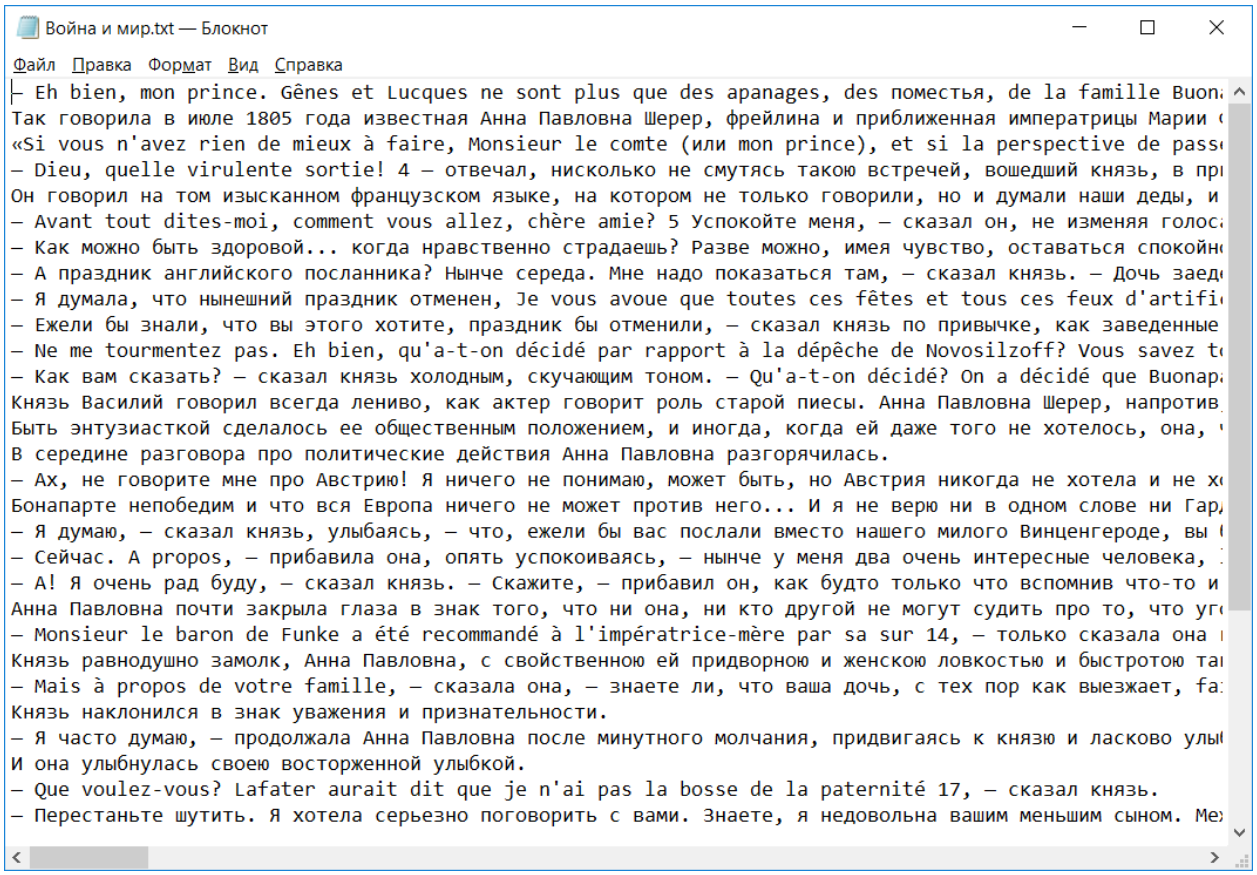

Рисунок 3 – Содержимое ключ-файла блокнота

Кратко об алгоритме шифрования: генерируется случайное число «a» от начала до середины ключ-файла блокнота. Начиная с этой позиции перебираются все следующие символы до первого совпадения символа шифруемого текста с символом блокнота. Когда совпадение происходит, в шифр-тест заносится порядковый номер символа в блокноте. Затем берется следующий символ исходного текста. Начиная с позиции совпадения предыдущего символа, ищется совпадение следующего. Если в процессе перебора символов блокнота был достигнут конец файла, то позиция «курсора» сместится на самое начало файла и продолжит движение, но не до конца файла, а до позиции, с которой начал перебор для текущего символа (то есть совпадения предыдущего).

Например, нам нужно зашифровать скороговорку (рисунок 4) на русском и языке и в ее переводе на английский. Усложним задачу, добавив в конце символ «?».

Рисунок 4 – Ввели текст для шифрования

Затем нажимаем кнопку шифрования (рисунок 5). Получаем массив чисел, каждый из которых соответствует порядковому номеру конкретного символа исходного текста в блокноте.

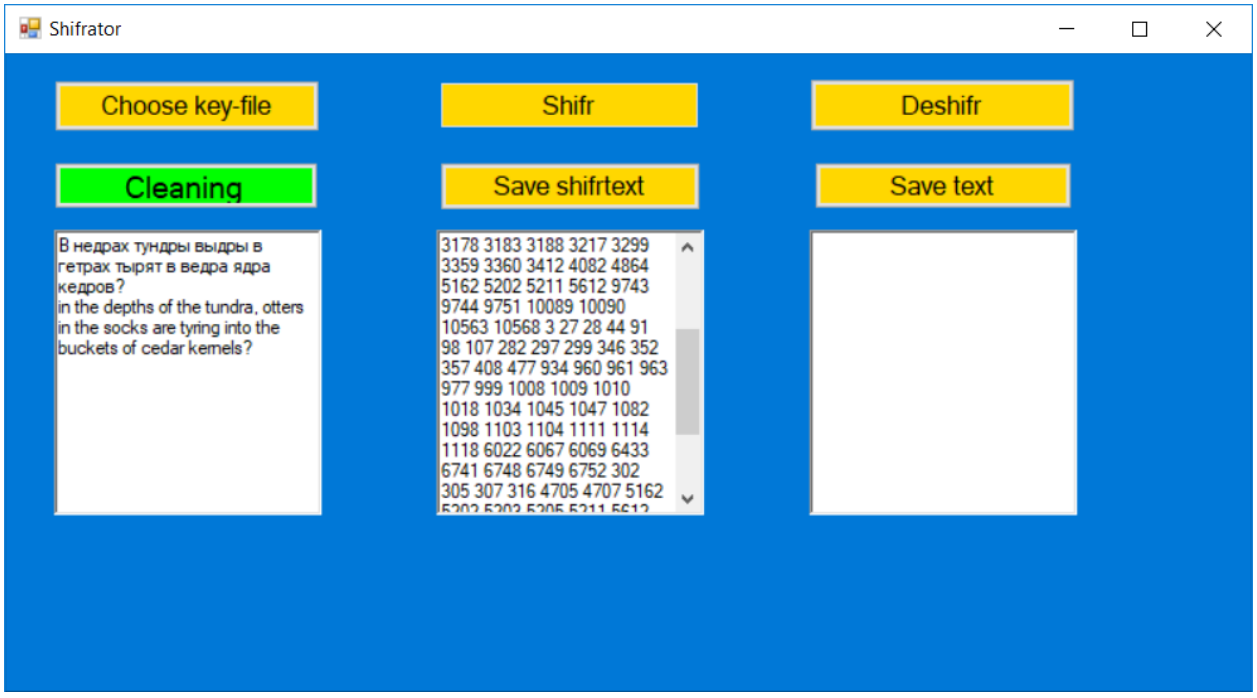

Рисунок 5 – Шифрование

Расшифруем, нажав кнопку дешифрования (рисунок 6)

Рисунок 6 – Расшифровывание

В данном примере мы в переводе скороговорки на английский использовали строчную “i” вместо прописной “I”. Попробуем поставить прописную и повторить операцию шифрования (рисунок 7).

Рисунок 7 – Отсутствие символа

Программа выводит сообщение о том, что во все файле-ключе нет такого символа, а значит, шифрование невозможно. Поставим этот символ самым первым в блокноте. Программа должна зашифровать его с номером 0 (рисунок 8).

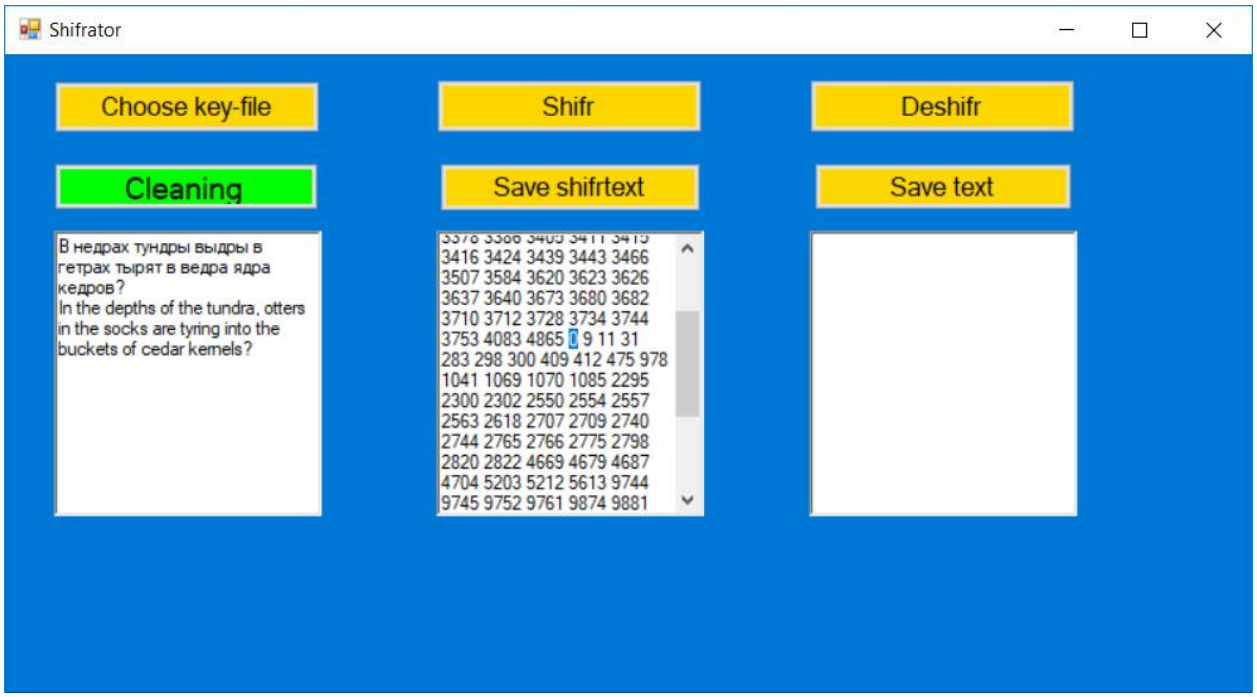

Рисунок 8 – Добавленный символ зашифровался

Кнопки открытия и сохранения файла вызывают проводник Windows, соответственно сохранение и открытие файлов происходят как при обычной работе с проводником (рисунки 9, 10).

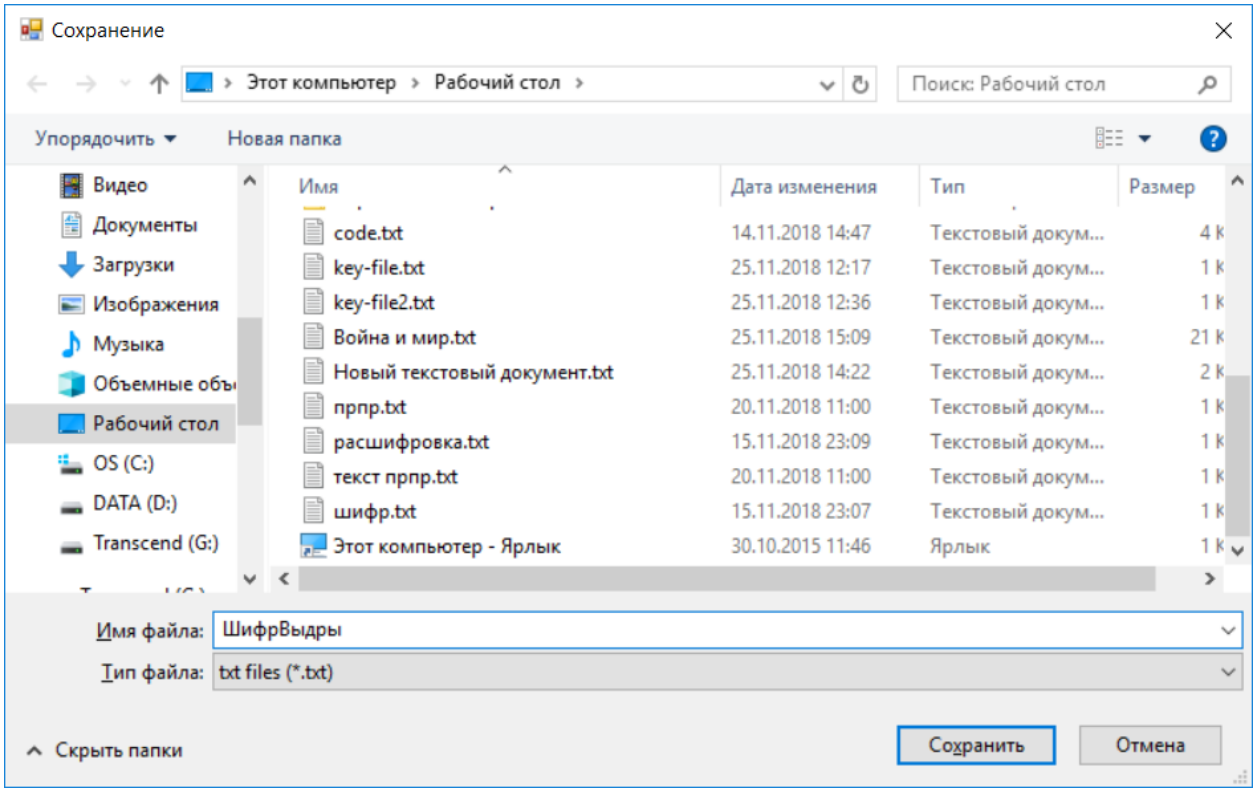

Рисунок 9 – Сохранение шифр-текста

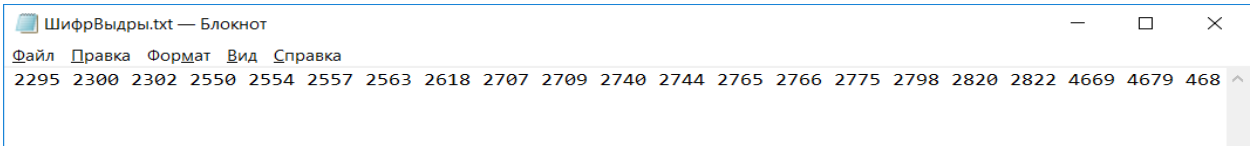

Рисунок 10 – Сохраненный шифр-текст

Ниже приведены листинги кодов для каждой кнопки, глобальные переменные, а также листинг кода, автоматически сгенерированного системой при создании формы (листинги 1-8). Программа написана на языке C++ как проект CLR.

Листинг 1 – Сгенерированный системой код, описывающий форму, и подключенные библиотеки

```
#pragma once
#include <cstdlib>
#include <math.h>

namespace Проект1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::IO;

	/// <summary>
	/// Сводка для Shifrator
	/// </summary>
	public ref class Shifrator : public System::Windows::Forms::Form
	{
	public:
		Shifrator(void)
		{
			InitializeComponent();
			//
			//TODO: добавьте код конструктора
			//
		}

	protected:
		/// <summary>
		/// Освободить все используемые ресурсы.
		/// </summary>
		~Shifrator()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::RichTextBox^  richTextBox1;
	protected:
	private: System::Windows::Forms::RichTextBox^  richTextBox2;
	private: System::Windows::Forms::RichTextBox^  richTextBox3;
```

```
            private: System::Windows::Forms::Button^  button1;
            private: System::Windows::Forms::Button^  button2;
            private: System::Windows::Forms::Button^  button3;

            private: System::Windows::Forms::Button^  button4;
            private: System::Windows::Forms::Button^  button5;
            private: System::Windows::Forms::Button^  button6;



            private:
                  /// <summary>
                  /// Обязательная переменная конструктора.
                  /// </summary>
                  System::ComponentModel::Container ^components;

      #pragma region Windows Form Designer generated code
                  /// <summary>
                  /// Требуемый метод для поддержки конструктора — не изменяйте
                  /// содержимое этого метода с помощью редактора кода.
                  /// </summary>
                  void InitializeComponent(void)
                  {
                        this->richTextBox1                =                (gcnew
System::Windows::Forms::RichTextBox());
                        this->richTextBox2                =                (gcnew
System::Windows::Forms::RichTextBox());
                        this->richTextBox3                =                (gcnew
System::Windows::Forms::RichTextBox());
                        this->button1 = (gcnew System::Windows::Forms::Button());
                        this->button2 = (gcnew System::Windows::Forms::Button());
                        this->button3 = (gcnew System::Windows::Forms::Button());
                        this->button4 = (gcnew System::Windows::Forms::Button());
                        this->button5 = (gcnew System::Windows::Forms::Button());
                        this->button6 = (gcnew System::Windows::Forms::Button());
                        this->SuspendLayout();
                        //
                        // richTextBox1
                        //
                        this->richTextBox1->Location                              =
System::Drawing::Point(366, 138);
                        this->richTextBox1->Name = L"richTextBox1";
                        this->richTextBox1->Size   =   System::Drawing::Size(225,
222);
                        this->richTextBox1->TabIndex = 0;
                        this->richTextBox1->Text = L"";
                        //
                        // richTextBox2
                        //
                        this->richTextBox2->Location                              =
System::Drawing::Point(681, 138);
                        this->richTextBox2->Name = L"richTextBox2";
                        this->richTextBox2->Size   =   System::Drawing::Size(225,
222);
                        this->richTextBox2->TabIndex = 1;
                        this->richTextBox2->Text = L"";
                        //
                        // richTextBox3
                        //
                        this->richTextBox3->Location = System::Drawing::Point(41,
138);
                        this->richTextBox3->Name = L"richTextBox3";
                        this->richTextBox3->Size   =   System::Drawing::Size(225,
222);
                        this->richTextBox3->TabIndex = 2;
                        this->richTextBox3->Text = L"";
                        //
                        // button1
```

```
                        //
                        this->button1->BackColor = System::Drawing::Color::Gold;
                        this->button1->Font               =               (gcnew
System::Drawing::Font(L"Microsoft           Sans           Serif",           12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                              static_cast<System::Byte>(204)));
                        this->button1->ForeColor = System::Drawing::Color::Black;
                        this->button1->Location = System::Drawing::Point(41, 21);
                        this->button1->Name = L"button1";
                        this->button1->Size = System::Drawing::Size(225, 41);
                        this->button1->TabIndex = 3;
                        this->button1->Text = L"Choose key-file";
                        this->button1->UseVisualStyleBackColor = false;
                        this->button1->Click += gcnew System::EventHandler(this,
&Shifrator::button1_Click);
                        //
                        // button2
                        //
                        this->button2->BackColor = System::Drawing::Color::Gold;
                        this->button2->Enabled = false;
                        this->button2->Font               =               (gcnew
System::Drawing::Font(L"Microsoft           Sans           Serif",           12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                              static_cast<System::Byte>(204)));
                        this->button2->ForeColor = System::Drawing::Color::Black;
                        this->button2->Location   =   System::Drawing::Point(366,
20);
                        this->button2->Name = L"button2";
                        this->button2->Size = System::Drawing::Size(225, 42);
                        this->button2->TabIndex = 4;
                        this->button2->Text = L"Shifr";
                        this->button2->UseVisualStyleBackColor = false;
                        this->button2->Click += gcnew System::EventHandler(this,
&Shifrator::button2_Click);
                        //
                        // button3
                        //
                        this->button3->BackColor = System::Drawing::Color::Gold;
                        this->button3->Enabled = false;
                        this->button3->Font               =               (gcnew
System::Drawing::Font(L"Microsoft           Sans           Serif",           12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                              static_cast<System::Byte>(204)));
                        this->button3->ForeColor = System::Drawing::Color::Black;
                        this->button3->Location   =   System::Drawing::Point(681,
20);
                        this->button3->Name = L"button3";
                        this->button3->Size = System::Drawing::Size(225, 42);
                        this->button3->TabIndex = 5;
                        this->button3->Text = L"Deshifr";
                        this->button3->UseVisualStyleBackColor = false;
                        this->button3->Click += gcnew System::EventHandler(this,
&Shifrator::button3_Click);
                        //
                        // button4
                        //
                        this->button4->BackColor = System::Drawing::Color::Gold;
                        this->button4->Font               =               (gcnew
System::Drawing::Font(L"Microsoft           Sans           Serif",           12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                              static_cast<System::Byte>(204)));
                        this->button4->ForeColor = System::Drawing::Color::Black;
                        this->button4->Location   =   System::Drawing::Point(368,
85);
                        this->button4->Name = L"button4";
                        this->button4->Size = System::Drawing::Size(222, 38);
                        this->button4->TabIndex = 7;
                        this->button4->Text = L"Save shifrtext";
                        this->button4->UseVisualStyleBackColor = false;
```

```
                        this->button4->Click += gcnew System::EventHandler(this,
&Shifrator::button4_Click);
                        //
                        // button5
                        //
                        this->button5->BackColor = System::Drawing::Color::Gold;
                        this->button5->FlatAppearance->BorderColor              =
System::Drawing::Color::Gold;
                        this->button5->FlatAppearance->BorderSize = 4;
                        this->button5->FlatAppearance->MouseDownBackColor       =
System::Drawing::Color::Red;
                        this->button5->FlatAppearance->MouseOverBackColor       =
System::Drawing::Color::Yellow;
                        this->button5->Font                =                (gcnew
System::Drawing::Font(L"Microsoft           Sans            Serif",            12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                              static_cast<System::Byte>(204)));
                        this->button5->ForeColor = System::Drawing::Color::Black;
                        this->button5->Location   =   System::Drawing::Point(686,
85);
                        this->button5->Name = L"button5";
                        this->button5->Size = System::Drawing::Size(219, 37);
                        this->button5->TabIndex = 8;
                        this->button5->Text = L"Save text";
                        this->button5->UseVisualStyleBackColor = false;
                        this->button5->Click += gcnew System::EventHandler(this,
&Shifrator::button5_Click);
                        //
                        // button6
                        //
                        this->button6->BackColor = System::Drawing::Color::Lime;
                        this->button6->Font                =                (gcnew
System::Drawing::Font(L"Microsoft           Sans            Serif",            14,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                              static_cast<System::Byte>(204)));
                        this->button6->Location = System::Drawing::Point(41, 85);
                        this->button6->Name = L"button6";
                        this->button6->Size = System::Drawing::Size(224, 37);
                        this->button6->TabIndex = 9;
                        this->button6->Text = L"Cleaning";
                        this->button6->UseVisualStyleBackColor = false;
                        this->button6->Click += gcnew System::EventHandler(this,
&Shifrator::button6_Click);
                        //
                        // Shifrator
                        //
                        this->AutoScaleDimensions   =   System::Drawing::SizeF(8,
16);
                        this->AutoScaleMode                                     =
System::Windows::Forms::AutoScaleMode::Font;
                        this->BackColor                                         =
System::Drawing::SystemColors::Highlight;
                        this->ClientSize = System::Drawing::Size(1056, 498);
                        this->Controls->Add(this->button6);
                        this->Controls->Add(this->button5);
                        this->Controls->Add(this->button4);
                        this->Controls->Add(this->button3);
                        this->Controls->Add(this->button2);
                        this->Controls->Add(this->button1);
                        this->Controls->Add(this->richTextBox3);
                        this->Controls->Add(this->richTextBox2);
                        this->Controls->Add(this->richTextBox1);
                        this->Name = L"Shifrator";
                        this->Text = L"Shifrator";
                        this->ResumeLayout(false);

                    }
```

Листинг 2 – Объявление области и глобальных переменных

```
      #pragma endregion

                  String^ key;
                  String^ ish;
                  String^ shifrtext;
                  String^ vspom;
                  String^ text;

                  int leng1;
                  int leng2;
                  unsigned int flag_r = 0;
```

Листинг 3 – Открытие файла-ключа

```
      //Открытие файла ключа
            private:   System::Void   button1_Click(System::Object^     sender,
System::EventArgs^  e) {
                  //Инициализируем диалог открытия окна
                  OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog;


                  //openFileDialog1->InitialDirectory = "c:\\";  //Начальный
путь
                  openFileDialog1->Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*"; //Предлагаемые форматы файлов
                  openFileDialog1->FilterIndex = 1;   //Тип файла по умолчанию
                  openFileDialog1->RestoreDirectory = true; //Откат  директории
на начало возможно....

                  if              (openFileDialog1->ShowDialog()              ==
System::Windows::Forms::DialogResult::OK) //Открываем оконце
                  {
                        StreamReader^ sr = gcnew  // выделение памяти
                        StreamReader(openFileDialog1->FileName,
System::Text::Encoding::GetEncoding(1251));
            //В открываемом файле учитываем кодировку RU языка
                        key = Convert::ToString(sr->ReadToEnd());
                        //кнопки 2 и 3 по умолчанию недоступны, активируем их
только после чтения файла-словаря

                  if ((key->Length) != 0)
                        {
                              this->button3->Enabled = true;
                              this->button2->Enabled = true;
                        }
                        sr->Close();
                                    }     }
```

Листинг 4 – Шифрование

```
                        //Шифруем
            private:   System::Void  button2_Click(System::Object^     sender,
System::EventArgs^  e) {
                  richTextBox1->Clear(); //очистка поля

                  ish = Convert::ToString(richTextBox3->Text);
      //чтение исходного текста
                  leng1 = key->Length;  //длина словаря
                  leng2 = ish->Length;   //длина исходного текста
                  int *vyh = new int[leng2];
      //массив выходных чисел шифртекста
                  int b = round(leng1 / 2);
                  int a = 0 + rand() % b; //сгенерированное случайное число
                  int z = 0;
      //ЗАПОМИНАЕТ ПОЗИЦИЮ ПОСЛЕДНЕГО НАЙДЕННОГО СИМВОЛА
                  for (int i = 0; i < leng2; i++) //перебор текста
                  {
                        flag_r = 0;
                        for (int j = 0; j < leng1; j++)  //перебор словаря
                        {

                              //с позиции случайного числа
                              if ((j + a + z) < leng1)

                              {
                                    if (ish[i] == key[j + a + z])
       //нашли совпадение
                                    {
                                          flag_r = 1;
      //символ успешно зашифрован
                                          vyh[i] = j + a + z;
       //текущая позиция курсора
                                          richTextBox1->Text                 +=
Convert::ToString(vyh[i]);
      //сразу выписываем число шифра
                                          richTextBox1->Text += " ";

                                          z = z + j + 1;
                                          if ((a + z) >= leng1)
                                                z = z - leng1;

                                          break;  //символ  зашифрован,  выходим
к следующей букве исходного текста
                                    }
                              }

                              else
                              {
                                    if  (ish[i]  ==  key[j  -  leng1  +  a  +  z])
//дошли до конца, идем с нуля до a
                                    {
                                          flag_r    =    1;    //символ    успешно
зашифрован
                                          vyh[i] = j - leng1 + a + z;
                                          richTextBox1->Text                 +=
Convert::ToString(vyh[i]);
                                          richTextBox1->Text += " ";

                                          z = z + j + 1;
                                          if ((a + z) >= leng1)
                                                z = z - leng1;
```

```
                                    break;
                              }
                        }

                  }

                  if (flag_r == 0)  //символ не был зашифрован
                  {
                        richTextBox1->Clear();
                        richTextBox1->Text   +=   "net   simvola   '"   +
Convert::ToString(ish[i]) + "'"; //сообщение о невозможности шифровки
                        break;
                  }
            }
            delete[] vyh;
      }
```

Листинг 5 – Дешифрование

```
       //расшифровываем
      private:   System::Void   button3_Click(System::Object^     sender,
System::EventArgs^  e) {
            richTextBox2->Clear();
            shifrtext = Convert::ToString(richTextBox1->Text);
            leng1 = key->Length;
            leng2 = shifrtext->Length;
            vspom = "";
            if (shifrtext[leng2 - 1] != ' ') //в конце нет пробела
            {
                  richTextBox1->Text    +=    Convert::ToString("    ");
//непорядок, пробел нужен, ставим
                  shifrtext   =   Convert::ToString(richTextBox1->Text);
//записываем обратно
                  leng2 = shifrtext->Length; //пересчитываем длину
            }
            //шифртекст в поле воспринимается как строка из-за пробелов,
                  //будем искать пробелы как ограничители и делить
строку на подтстроки без пробелов
                  //которые потом конвертируем в целочисленное
            int z = 0;
            for (int i = 0; i < leng2; i++) //по длине шифр текста в
символах
            {


                  if (shifrtext[i] != ' ')
                  {
                        vspom += shifrtext[i]; //маленькая строка
с числом

                  }
                  else
                  {
                        z = Convert::ToInt32(vspom);//делаем целое
число, раз дошли до пробела
                        richTextBox2->Text                        +=
Convert::ToString(key[z]); //выписываем символ словаря с указанным номером
                        vspom = ""; //обнулили строку
                  }
```

```
            }
        }
```

Листинг 6 – Сохранение шифр-текста в файл

```
        private:   System::Void   button4_Click(System::Object^     sender,
System::EventArgs^  e) {
                //Сохранение результата шифр-текста в файл
                using namespace System::IO;
                //Инициализируем диалог открытия окна
                SaveFileDialog^ saveFileDialog1 = gcnew SaveFileDialog;
                saveFileDialog1->Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
                saveFileDialog1->FilterIndex = 1; //по умолчание txt
                saveFileDialog1->RestoreDirectory = true;

                if             (saveFileDialog1->ShowDialog()             ==
System::Windows::Forms::DialogResult::OK)
                {
                      StreamWriter^ wr = gcnew
                            StreamWriter(saveFileDialog1->FileName,    false,
System::Text::Encoding::GetEncoding(1251));
                      wr->Write(richTextBox1->Text);
                      wr->Close();
                }
        }
```

Листинг 7 – Сохранение расшифрованного текста в файл

```
    private:    System::Void    button5_Click(System::Object^        sender,
System::EventArgs^  e) {
        //Сохранение результата расшифрованного текста в файл
        using namespace System::IO;
        SaveFileDialog^ saveFileDialog1 = gcnew SaveFileDialog;
        saveFileDialog1->Filter  =  "txt  files  (*.txt)|*.txt|All  files
(*.*)|*.*";
        saveFileDialog1->FilterIndex = 1; //по умолчанию txt
        saveFileDialog1->RestoreDirectory = true; //запоминание директории

        if                (saveFileDialog1->ShowDialog()                ==
System::Windows::Forms::DialogResult::OK)
        {
              StreamWriter^ wr = gcnew
                    StreamWriter(saveFileDialog1->FileName,          false,
System::Text::Encoding::GetEncoding(1251));
              wr->Write(richTextBox2->Text);
              wr->Close();
        }
    }
```

Листинг 8 – Очистка полей

```
      private:    System::Void    button6_Click(System::Object^         sender,
System::EventArgs^  e) {
            richTextBox1->Clear();
            richTextBox2->Clear();
            richTextBox3->Clear();
      }
```

Выводы: На языке C++ было создано приложение CLR, позволяющее шифровать и дешифровать текст методом «Блокнот». Метод обладает достаточно высокой криптостойкостью благодаря тому, что один и тот же символ шифруется разными числами, и только при наличии файла-ключа можно выявить соответствие числа символу. В отличие от шифрования методом «шифр Цезаря», например, где выявить соответствие символа коду можно опираясь на знание частоты появление символа в языке. К недостаткам можно отнести то, что файл-ключ обязательно должен быть длинным и содержать все символы, которые необходимо зашифровать в исходном тексте.