

**Цель работы:**  
Необходимо изучить ключевые существующие методы и алгоритмы криптографической защиты информации. Продемонстрировать полученные знания при выполнении лабораторной работы путем представления умений и навыков работы с рассматриваемыми методами и алгоритмами на примере криптографического алгоритма/протокола шифрования и/или криптографического алгоритма сокрытия информации.

**Варианты выполнения работы:**

1) Без использования СВТ

- выбирается два протокола шифрования (ассиметричный и симметричный). Описывается и демонстрируется их работа во время занятия.

2) С использованием СВТ:

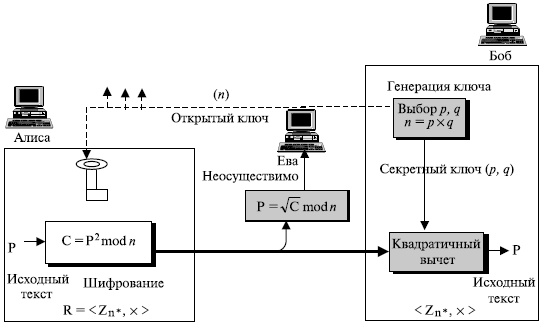
Написать программную реализацию двух криптографических протоколов (шифрования/сокрытия информации) не входящих в одну типовую группу.

P.S. Нельзя использовать алгоритмы: Цезарь и RSA.

Были выбраны: **Криптосистема Рабина** и **Tiny Encryption Algorithm**

**Теоретические сведения**

**Криптосистема Рабина** — [криптографическая система с открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), безопасность которой обеспечивается [сложностью](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) поиска [квадратных корней](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D1%80%D0%B5%D0%BD%D1%8C) в [кольце](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%8C%D1%86%D0%BE_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) [остатков по модулю](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E) [составного числа](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE)



**Генерация ключа**

Система Рабина, как и любая [асимметричная криптосистема](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), использует открытый и закрытый ключи. Открытый ключ используется для [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) сообщений и может быть опубликован для всеобщего обозрения. Закрытый ключ необходим для расшифровки и должен быть известен только получателям зашифрованных сообщений.

Процесс генерации ключей следующий:

* выбираются два случайных числа *p* и *q* с учётом следующих требований:
  + числа должны быть большими (см. [разрядность](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D1%8F%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%8C));
  + числа должны быть [простыми](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE);
  + p mod 4 = 3
  + q mod 4 = 3

Выполнение этих требований сильно ускоряет процедуру извлечения корней по модулю *р* и *q*;

* вычисляется число *n* = *p* · *q*;

Число *n* — открытый ключ; числа *p* и *q* — закрытый.

**Шифрование**

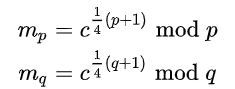
Исходное сообщение *m* (текст) шифруется с помощью открытого ключа — числа *n* по следующей формуле:

*c* = *m*² mod *n*.

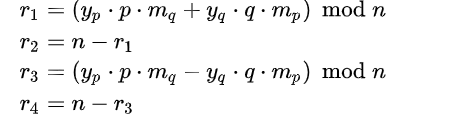
**Расшифрование**

Для расшифровки сообщения необходим закрытый ключ — числа *p* и *q*. Процесс расшифровки выглядит следующим образом:

* сначала, используя расширенный алгоритм Евклида, из уравнения Yp \* p +Yq\*q=1 найти значения Yp,Yq
* Вычислите квадратный корень по модулю:

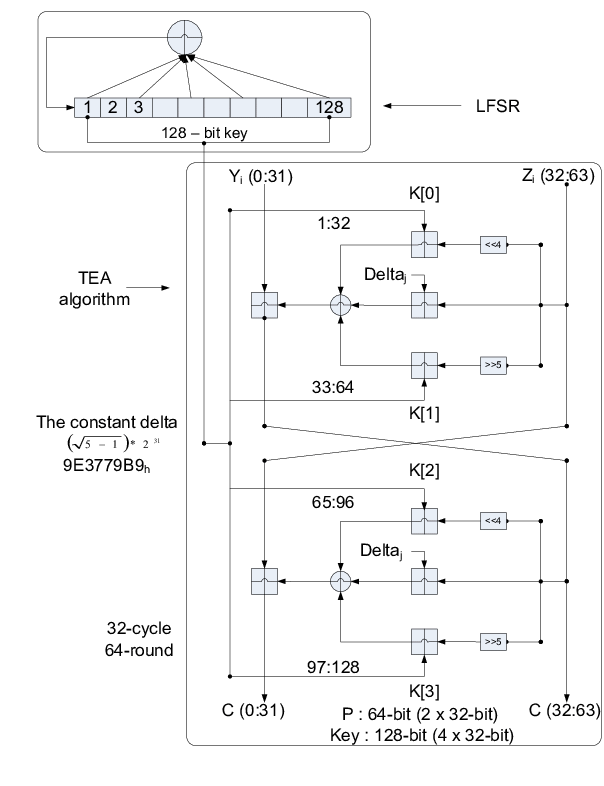


* Далее, используя [китайскую теорему об остатках](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D1%82%D0%B0%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%BE%D0%B1_%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B0%D1%85), вычисляют четыре числа:



Одно из этих чисел является истинным открытым текстом *m*.

**Tiny Encryption Algorithm (TEA)** — блочный алгоритм шифрования типа «Сеть Фейстеля». Шифр не патентован, широко используется в ряде криптографических приложений и широком спектре аппаратного обеспечения благодаря крайне низким требованиям к памяти и простоте реализации.



**Описание алгоритма**

Исходный текст разбивается на блоки по 64 бита каждый.   
128-битный ключ К делится на четыре 32-битных подключа K[0], K[1], K[2] и K[3].

На этом подготовительный процесс заканчивается, после чего каждый 64-битный блок шифруется на протяжении 32 циклов (64 раундов) по нижеприведённому алгоритму.

Предположим, что на вход n-го раунда (для 1 ≤ n ≤ 64) поступают правая и левая часть (Ln, Rn), тогда на выходе n-го раунда будут левая и правая части (Ln+1, Rn+1), которые вычисляются по следующим правилам:

Ln+1 = Rn.  
Если раунд нечётный:

Если раунд четный:  


Где

  
Операция сложения чисел X и Y по модулю 232.

  
побитовое исключающее «ИЛИ» (XOR) чисел X и Y

  
операции побитового сдвига числа X на Y бит влево и вправо соответственно

Константа δ была выведена из Золотого сечения δ =  = 2654435769 = 9E3779B9. В каждом раунде константа умножается на номер цикла i. Это было сделано для предотвращения простых атак, основанных на симметрии раундов

Также очевидно, что в алгоритме шифрования TEA нет как такового алгоритма расписания ключей. Вместо этого в нечётных раундах используются подключи К[0] и К[1], в чётных — К[2] и К[3].

Так как это блочный шифроалгоритм, где длина блока 64-бит, а длина данных может быть не кратна 64-битам, значения всех байтов дополняющих блок до кратности в 64-бит устанавливается в 0x01 .

**Исходный код программы:**

Файл Rabin.py

#Рабин

import random

import math

import re

from Crypto.Util import number

#def primes(num):

# if 2 <= num:

# yield 2

# for i in range(3, num + 1, 2):

# if all(i % x != 0 for x in range(3, int(math.sqrt(i) + 1))):

# yield i

#

##На больших числа тупит

#def big\_mod\_prime(before,size):#Бефор-до какого числа генерируется значения(правая граница).Сайз-левая граница

# z = list(primes(before))

# print(z)

# z = [x for x in z if x > size]#Оставляет ток элементы выше size

# z = [x for x in z if (x % 4) == 3]#Оставляет только те что z mod 4 = 3

# position = random.randint(0, len(z) -1)

# return z[position]

#генерация простых чисел (больший и с модом)

def fast\_big\_mod\_prime(len):

primeNum = 0

while (primeNum % 4 != 3):

primeNum = number.getPrime(len)

return primeNum

#def flatten(l):

# for el in l:

# if isinstance(el, collections.Iterable) and not isinstance(el, (str, bytes)):

# yield from flatten(el)

# else:

# yield el

#

def ASCIItoText(number):

number = str(number)

size = len(number)

if (size % 2 != 0 ):

number = "0" + number

number = [number[i:i+2] for i in range(0, len(number),2)]

# number = (str(([ chr( int(number[i]) ) for i in range( 0, len( number ) )])))

number = (([ chr( int(number[i]) ) for i in range( 0, len( number ) )]))

return (''.join(number))

#def ev(a,b):

# c = 1 #Нод

# for k in range(abs(a)):

# if ( c - b \* k ) % a == 0:

# y = k

# x = ( c - b \* y ) // a

# return x,y

#

def bezout\_recursive(a, b):

if not b:

return (1, 0, a)

y, x, g = bezout\_recursive(b, a % b)

return (x, y - (a // b) \* x, g)

#размер NORM OT 100 DO 1000

size\_bit = 100

q = fast\_big\_mod\_prime(size\_bit )

p = fast\_big\_mod\_prime(size\_bit)

print("Закрытые случайные ключи")

print(f"q = {q}")

print(f"p = {p}")

n = q\*p

print("Открытый ключ n -", n)

#Создаем сообщение М<n

#M = int(input('Введите M:'))

M\_i= str(input('\nПожалуйста,Введите сообщение:'))

print("\nИсходное сообщение:", M\_i)

M\_i =[ord(c) for c in M\_i]

#print("Исходное сообщение:", M\_i)

M =" "

for i in M\_i:

M += str(i)

M = int(M)

print("Исходное сообщение (M) :", M)

#Шифрование c =N^2 % n

C = pow(M, 2, n)

print('\nЗашифрованное сообщение(C):', C)

#Расшифровка

mp = pow(C, (p+1)//4, p)# c^((p+1)/4) mod p

mq = pow(C, ((q+1)//4), q)

#Расширенный алгоритм Eвклида Yp\*p+Yq \*q =1

yp,yq,g = bezout\_recursive(p,q)

#вычислим a и b

M1 = (yp\*p\*mq + yq \* q \* mp) % n

M2 = n - M1

M3 =(yp\*p\*mq - yq \* q \* mp) % n

M4 = n - M3

print("\nРасшифровки")

print("Вариат 1=",M1)

print("Вариат 2=",M2)

print("Вариат 3=",M3)

print("Вариат 4=",M4)

print("\nТекстовое представление расшифрованных сообщений")

print(f"Вариат 1 = {ASCIItoText(M1)} \n")

print(f"Вариат 2 = {ASCIItoText(M2)} \n")

print(f"Вариат 3 = {ASCIItoText(M3)} \n")

print(f"Вариат 4 = {ASCIItoText(M4)} \n")

Файл tea1.cpp

#include <iostream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <fstream>

#include <cstddef>

#include <vector>

#include <map>

#include <array>

using namespace std;

//Сеть Фейстеля

void Feistel(unsigned int\* v, unsigned int\* w, unsigned int\* k) {

unsigned int v0 = v[0], v1 = v[1], i, sum = 0;

unsigned int delta = 0x9E3779B9;//Константа δ

//РАУДОВ 64,НО ЭТО 32 ЦИКЛА (2 РАУНДА В 1 ЦИКЛЕ)

for (i = 0; i < 32; i++) {

cout << "\nНомер цикла в сете Фейстеля=" << i << endl;

sum += delta;

v0 += ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);//НЕЧЕТНЫЕ РАУНДЫ

v1 += ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);//ЧЕТНЫЕ РАУНДЫ

cout << "L(левая часть блока)=" << v0 << endl;

cout << "R(правая часть блока)=" << v0 << endl;

}

w[0] = v0; w[1] = v1;

}

void TeaEncode(const std::string& str, const std::string& key, std::string\* out)

{

unsigned int v[2];

unsigned int w[2];

unsigned int k[4];

unsigned int keybuffer[4];

//memset - заполняет массива (1 параметр) элементами(2 пармаметр) длингой (3 параметр)

memset(v, 0, sizeof(v));

memset(w, 0, sizeof(w));

memset(k, 0, sizeof(k));

memset(keybuffer, 0, sizeof(keybuffer));

out->clear();

int len = key.length();

if (len > 16)

len = 16;

//Копирует байты от ключа в ключ\_буфера

memcpy(keybuffer, key.c\_str(), len);

for (int i = 0; i < 4; ++i) {

k[i] = keybuffer[i];

}

//Скопируйте входную строку в буфер размером, кратным 4

int strbuflen = str.length();

if (strbuflen == 0)

return;

if ((strbuflen % 4) > 0)

strbuflen += 4 - (strbuflen % 4);

unsigned char\* strbuf = new unsigned char[strbuflen];

// сначала очищается а потом как заполняется

memset(strbuf, 0, strbuflen);

memcpy(strbuf, str.c\_str(), str.length());

v[1] = 0;

for (int i = 0; i < strbuflen; i += 4)

{

v[0] = \*(unsigned int\*)&strbuf[i];

Feistel(&v[0], &w[0], &k[0]);

out->append((char\*)&w[0], 4);

v[1] = w[1];

}

out->append((char\*)&v[1], 4);

delete[] strbuf;

}

int main()

{

setlocale(LC\_ALL, "Russian");

ifstream fileIn;

ofstream fileOut;

string f1, f2, f3;

string\* fileName = &f1; // имя файла

string\* key = &f2; // ключ

string\* data = &f3; // строка

string\* name = new string;

cout << "Введите ключ=";// << endl;

getline(cin, \*key);

cout << "Введите название файла=";// << endl;

getline(cin, \*fileName);

fileIn.open(\*fileName, ios::in | ios::binary);

string str((istreambuf\_iterator<char>(fileIn)), istreambuf\_iterator<char>());

fileIn.close();

string outdata;

cout << "Поступившая строка=" << str << " Поступивший ключ" << f2 << endl;

TeaEncode(str, f2, &outdata);

str = "0";

cout << "Введите название файла для сохранения результатов=";// << endl;

getline(cin, \*fileName);

fileOut.open(\*fileName, ios::out | ios::binary);

fileOut.write(outdata.c\_str(), outdata.size());

// End

cout << "Пока!" << endl;

system("pause");

}

Файл tea2.cpp

#include <iostream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <fstream>

#include <cstddef>

#include <vector>

#include <map>

#include <array>

using namespace std;

void Feistel(unsigned int\* v, unsigned int\* w, unsigned int\* k) {

register unsigned int v0 = v[0], v1 = v[1], i, sum = 0xC6EF3720;

register unsigned int delta = 0x9E3779B9;

for (i = 0; i < 32; i++) {

cout << "\nНомер цикла в сете Фейстеля=" << i << endl;

v1 -= ((v0 << 4) + k[2]) ^ (v0 + sum) ^ ((v0 >> 5) + k[3]);

v0 -= ((v1 << 4) + k[0]) ^ (v1 + sum) ^ ((v1 >> 5) + k[1]);

sum -= delta;

cout << "L(левая часть блока)=" << v0 << endl;

cout << "R(правая часть блока)=" << v0 << endl;

}

w[0] = v0; w[1] = v1;

}

void TeaDecode(const std::string& str, const std::string& key, std::string\* out)

{

unsigned int v[2];

unsigned int w[2];

unsigned int k[4];

unsigned int keybuffer[4];

memset(v, 0, sizeof(v));

memset(w, 0, sizeof(w));

memset(k, 0, sizeof(k));

memset(keybuffer, 0, sizeof(keybuffer));

out->clear();

int numBlocks = str.length() / 4;

int numPasses = numBlocks - 1;

if (numPasses <= 0)

return;

int len = key.length();

if (len > 16)

len = 16;

memcpy(keybuffer, key.c\_str(), len);

for (int i = 0; i < 4; ++i)

k[i] = keybuffer[i];

unsigned char\* buffer = new unsigned char[numPasses \* 4 + 4];

memset(buffer, 0, numPasses \* 4 + 4);

const char\* p = str.c\_str();

v[1] = \*(unsigned int\*)&p[numPasses \* 4];

for (int i = 0; i < numPasses; ++i)

{

v[0] = \*(unsigned int\*)&p[(numPasses - i - 1) \* 4];

Feistel(&v[0], &w[0], &k[0]);

\*(unsigned int\*)&buffer[(numPasses - i - 1) \* 4] = w[0];

v[1] = w[1];

}

out->assign((char\*)buffer, numPasses \* 4);

delete[] buffer;

}

int main()

{

setlocale(LC\_ALL, "Russian");

ifstream fileIn;

ofstream fileOut;

string f1, f2, f3;

string\* fileName = &f1;

string\* key = &f2;

string\* data = &f3;

string\* name = new string;

cout << "Введите ключ=";// << endl;

getline(cin, \*key);

cout << "Введите название файла=";// << endl;

getline(cin, \*fileName);

fileIn.open(\*fileName, ios::in | ios::binary);

string str((istreambuf\_iterator<char>(fileIn)), istreambuf\_iterator<char>());

fileIn.close();

string outdata;

cout << "Поступившее сооющение=" <<str<< "Поступивший ключ=" << f2<< endl;

TeaDecode(str, f2, &outdata);

str = "0";

cout << "Введите название файла для сохранения результатов=";// << endl;

getline(cin, \*fileName);

fileOut.open(\*fileName, ios::out | ios::binary);

fileOut.write(outdata.c\_str(), outdata.size());

cout << "Пока!" << endl;

system("pause");

}

**Скриншоты:**

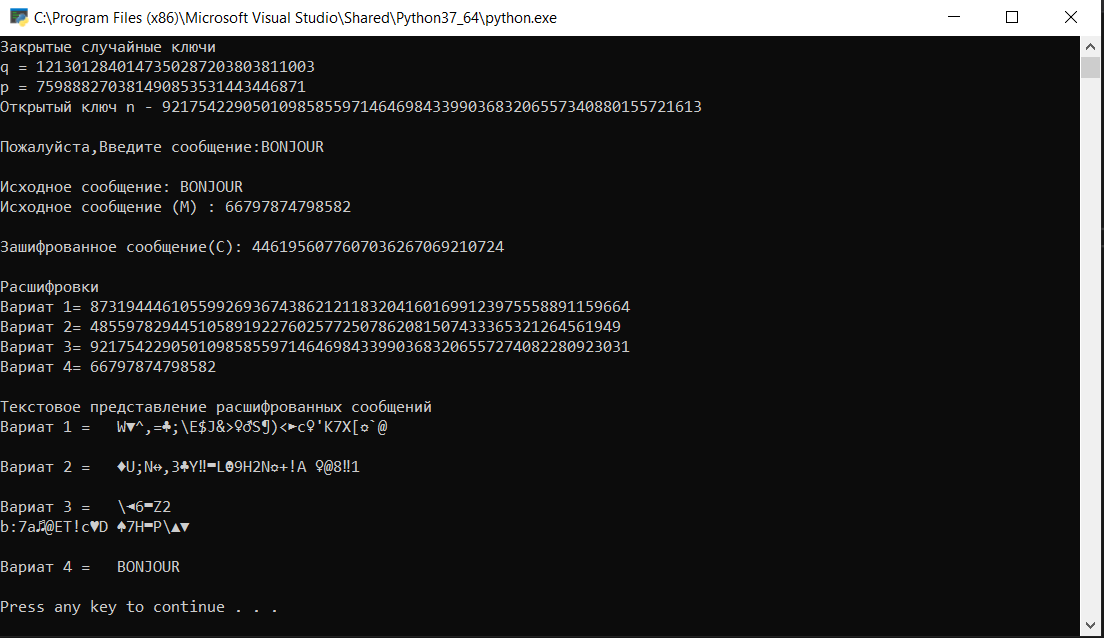


Рисунок 1-работа криптосистемы Рабина

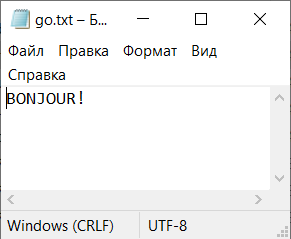


Рисунок 2-текстовый файл go.txt для алгоритм tea(исходное сообщение )

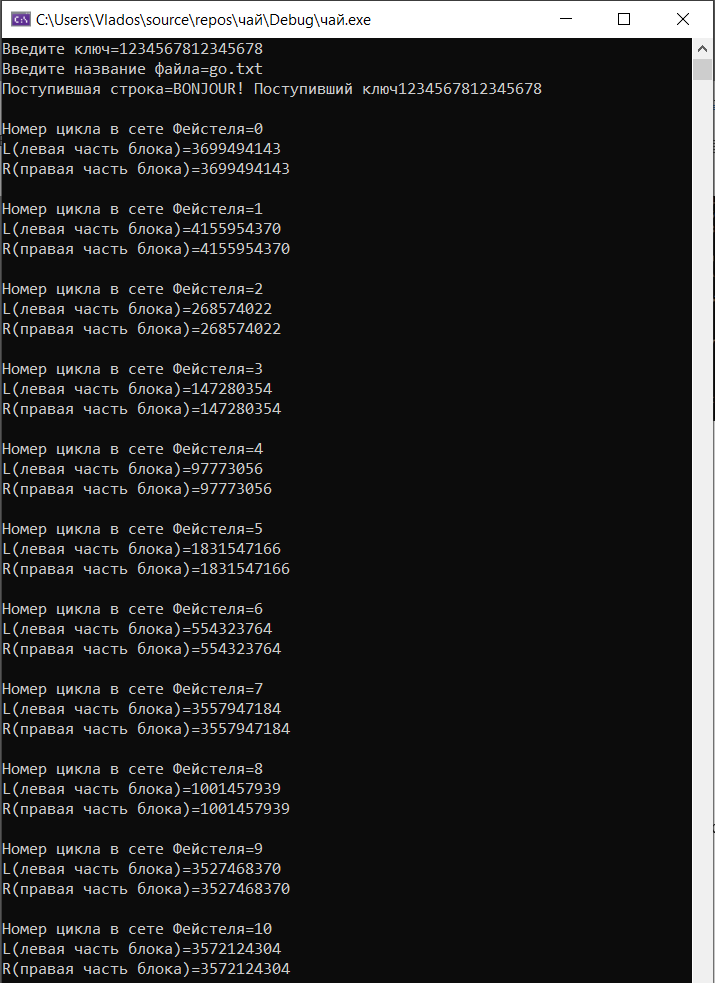


Рисунок 3- работа алгоритма TEA для шифрования текста (представлены первые 11 циклов сети Фейстеля)

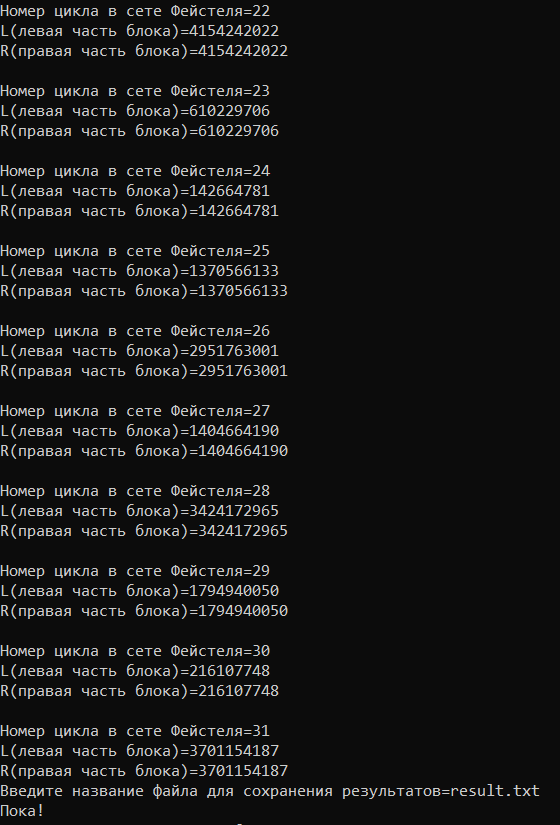


Рисунок 4- работа алгоритма TEA для шифрования текста (циклы сети Фейстеля с 23 по 32)  
Создан файл result.txt с зашифроваванным сообщением

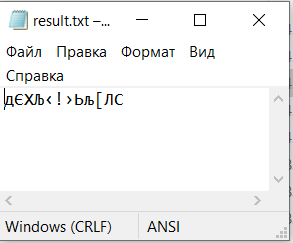
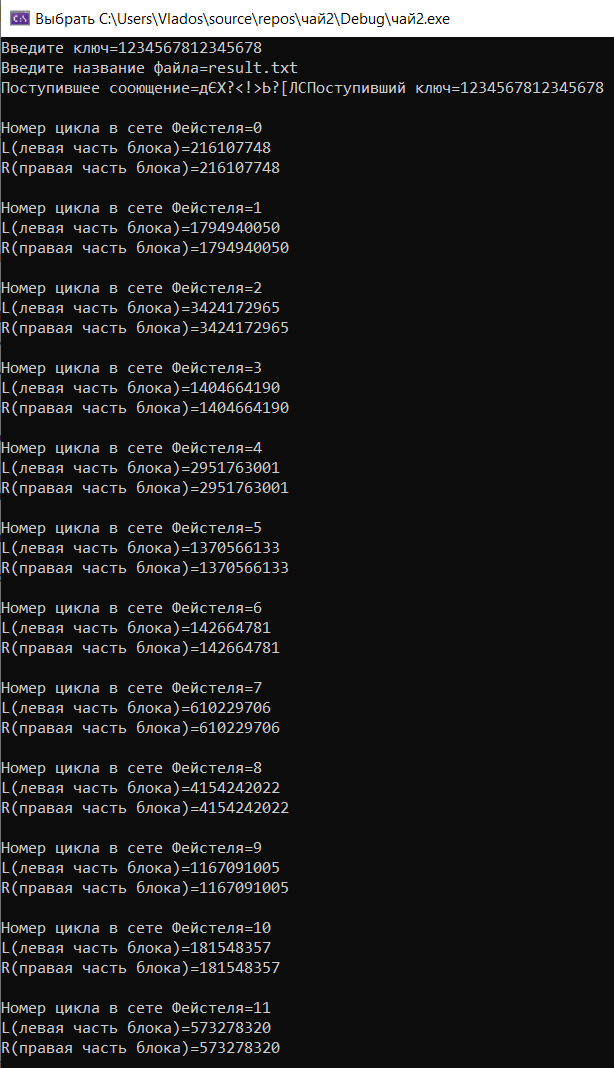


Рисунок 5-текстовый файл result.txt для алгоритм tea(зашифрованное сообщение )

  
Рисунок 6- работа алгоритма TEA для дешифрования текста (представлены первые 12 циклов)

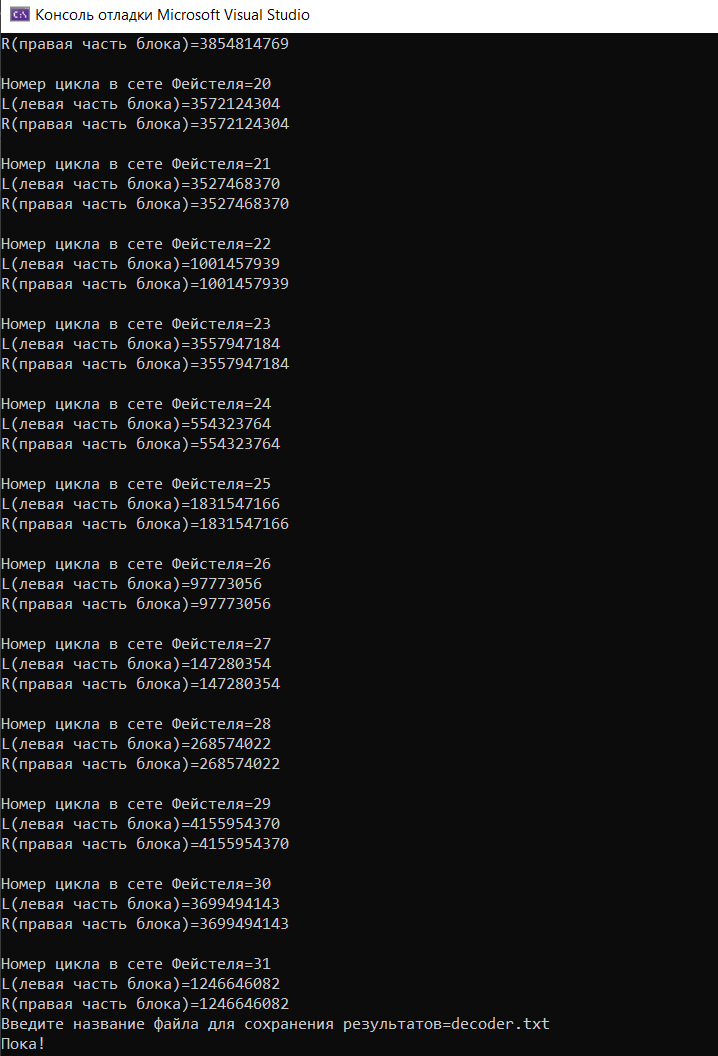


Рисунок 7- работа алгоритма TEA для дешифрования текста (представлены циклы сети Фейстеля с 21 по 32)

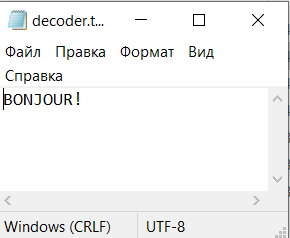


Рисунок 8-текстовый файл decoder.txt для алгоритм tea(дешифрованное сообщение )

**Выводы**

В ходе лабораторной работы были изучены ключевые существующие методы и алгоритмы криптографической защиты информации. Продемонстрированы полученные знания при выполнении лабораторной работы путем представления умений и навыков работы с рассматриваемыми методами и алгоритмами на примере криптографического алгоритма/протокола шифрования и/или криптографического алгоритма сокрытия информации.