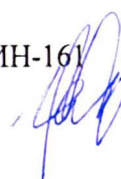



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
«Омский государственный технический университет»  
Кафедра «Автоматизированные системы обработки информации и  
управления»

## ОТЧЕТ К ДОМАШНЕЙ РАБОТЕ

По дисциплине «Защита информации»

Выполнил:  
Студент группы ПИН-161  
Леммер В.Е.  
Проверил:  
Доцент  
Щелканов А.В.   
 19.12.19

# ДОМАШНЕЕ ЗАДАНИЕ

## РЕАЛИЗАЦИЯ АЛГОРИТМОВ ШИФРОВАНИЯ

*Цель работы:* Изучение и реализация принципов работы алгоритмов шифрования Цезаря, Аффинного шифра, Виженера, методом гаммирования, сетями Фейстеля, RSA

### 1. Теоретические сведения

#### Шифр Цезаря

Строки таблицы подстановки для шифра Цезаря представляют собой сдвинутые друг относительно друга на  $n$  позиций алфавиты исходного текста. Сам Цезарь использовал для шифрования величину сдвига  $n = 3$ .

Функция шифрования моноалфавитного шифра имеет вид:

$$E_n(x) = (x + n) \bmod m,$$

где  $x$  – числовой код буквы открытого текста;  $n$  – ключ, определяющий величину циклического сдвига;  $m$  – мощность входного алфавита (количество букв в используемом алфавите). Таким образом, ключом шифрования здесь является число  $k$ , определяющее размер смещения. Расшифровка осуществляется аналогично:

$$D_n(x) = (x - n) \bmod m$$

#### Аффинный шифр

Шифр Цезаря является частным случаем аффинного шифра. Функция шифрования аффинного шифра имеет вид:

$$E_n(x) = (ax + b) \bmod m,$$

где пара  $a$  и  $b$  – ключ шифра. Значение  $a$  должно быть выбрано таким, что  $a$  и  $m$  – взаимно простые числа.

Функция расшифрования:

$$D_n(x) = a^{-1}(x - b) \bmod m,$$

где  $a^{-1}$  – обратное к  $a$  число по модулю  $m$ .

#### Шифр Виженера

Каждая буква ключа определяет свой алфавит шифрования, который получается из нормативного циклическим сдвигом на количество символов, равное числовому эквиваленту буквы ключа. Очевидно, что длина ключа равна периоду шифра. Чтобы зашифровать сообщение шифром Виженера, поступают следующим образом. Под каждой буквой открытого текста помещается буква ключа. Ключ циклически повторяется необходимое число раз.

Для расшифровки нужно найти длину ключа и сам ключ, с помощью индекса совпадений и частотного анализа.

Метод основывается на вычислении вероятности того, что два случайных элемента текста совпадут. Эту вероятность называют индексом совпадений. Уильям Фридман показал, что значения индекса совпадений существенно отличаются для текстов различной природы. Это позволяет сначала определить длину ключа шифра, а затем найти и сам ключ.

Индекс совпадений вычисляется по формуле:

$$ИС = \frac{\sum_{i=1}^L f_i(f_i - 1)}{n(n - 1)}$$

где  $f_i$  – количество появлений  $i$ -ой буквы алфавита в тексте;  $L$  – мощность алфавита;  $n$  – количество символов в криптограмме.

Для английского языка индекс совпадений имеет значение 0.0667, в то время как для случайного набора букв этот показатель равен 0.038.

### **Метод гаммированием**

В основе рассматриваемых систем шифрования лежит метод «наложения» ключевой последовательности – гаммы – на открытый текст. «Наложение» заключается в позначном (побуквенном) сложении или вычитании по тому или иному модулю. Обычно берется сложение по модулю два или по модулю  $N$  ( $N$  – число символов алфавита открытого текста). В силу простоты своей технической реализации и высоких криптографических качеств эти шифры получили широкое распространение.

Принцип шифрования гаммированием заключается в генерации гаммы шифра с помощью датчика псевдослучайных чисел и наложении полученной гаммы на открытые данные обратимым образом. Процесс дешифрования данных сводится к повторной генерации гаммы шифра при известном ключе и наложении такой гаммы на зашифрованные данные.

### **Сети Фейстеля**

Сетью Фейстеля называется метод обратимых преобразований текста, при котором значение, вычисленное от одной из частей текста, накладывается на другие части. Часто структура сети выполняется таким образом, что для шифрования и дешифрования используется один и тот же алгоритм – различие состоит только в порядке использования материала ключа.

Независимые потоки информации, порождённые из исходного блока, называются ветвями сети. Величины  $V_i$  именуются параметрами сети, обычно это функции от материала ключа. Функция  $F$  называется образующей. Действие, состоящее из однократного вычисления образующей функции и последующего наложения её результата на другую ветвь с обменом их местами, называется циклом или раундом сети Фейстеля.

### **RSA**

Наиболее широко используемым асимметричным алгоритмом в настоящее время является RSA. Рон Ривест, Ади Шамир и Леонард Адлеман (Ron Rivest, Adi Shamir, Leonard Adleman) изобрели шифр RSA в 1978 году в ответ на идеи, выдвинутые

Хеллманом, Диффи и Меркелем (Heilman, Diffie, Merkel). Рассмотрим RSA на концептуальном уровне.

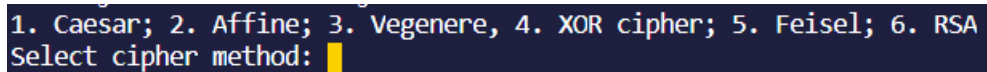
Вначале мы генерируем случайную пару ключей. Как и вообще, в криптографии очень важно сгенерировать ключи самым случайным и, следовательно, самым непредсказуемым способом. Затем мы шифруем данные открытым ключом при помощи алгоритма RSA. Наконец, мы дешифруем сообщение при помощи секретного ключа и убеждаемся в совпадении полученного результата с исходным сообщением.

## 2. Интерфейс и работа с программой

Все алгоритмы шифрования были реализованы на объектно-ориентированном языке программирования C++. Для сборки составных частей проекта include, src и res файлов использовалась система проектной сборки CMake.

Программа считывает исходный алфавит и текст с файла, для этого был реализован класс `file_manager`. Дополнительные функции, такие как удаление пробелов, нахождение определенной буквы в алфавите и так далее, находится в классе `service_manager`. Основной код программы представлен в приложении листинг 1 – 9.

Взаимодействие через программу происходит через консоль. (рис. 1) Пользователей вводит цифру нужного ему метода и дальше работает с ним, например вводит ключ шифрования в цифровом или текстовом виде.



```
1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: █
```

Рисунок 1 – Консоль программы

Проведем тестирование методов шифрования.

Ключ шифрования для Цезаря: 4;

Ключ шифрования для Аффинного шифра: 21 и 5;

Текстовый ключ шифрования: `skey`;

Ключ Фейстеля: 5;

Исходный алфавит: `abcdefghijklmnopqrstuvwxyz`;

Исходный текст: *Maecenas lacinia eu lorem quis vulputate. Nam congue, ante nec tempor rutrum, ex lectus vestibulum felis, sit amet feugiat quam mauris quis dui. Fusce nulla justo, mattis vel augue sed, ornare dapibus nisi. Etiam consectetur, elit vel bibendum tincidunt, mi ex blandit urna, at posuere nisi mi at purus. Proin eu erat magna. Aliquam ac lectus nulla. Cras pretium orci dolor. Pellentesque quis odio augue. Sed dignissim vestibulum ullamcorper. Vestibulum facilisis ligula eget dolor tincidunt, in mollis dui ultrices. Vestibulum euismod dolor lorem, et accumsan arcu interdum pulvinar.*

```

1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: 1

Enter key (number): 4

Encrypt: qeigirew pegmrme iy psviq uymw zyptyxexi. req gsrkyi, erxi rig xiqtsv vyxvyq, ib
pigxyw ziwxmfypq jipmw, wmx eqix jiykmex uyeq qeyvmw uymw hym. jywgi ryppe nywxs, qexxmw
zip eykyi wih, svrevi hetmfyw rmwm. ixmeq gsrwigxixyv, ipmx zip fmfirhyq xmrgmhyrx, qm ib
fperhmx yvre, ex tsyivi rmwm qm ex tyvyw. tvsmr iy ivex qekre. epmuyeq eg pigxyw ryppe. g
vew tvixmyq svgm hspsv. tippirxiwuyi uymw shms eykyi. wih hmkrmwmq ziwxmfypq yppeqgsvtiv
. ziwxmfypq jegmpmwmw pmkype ikix hspsv xmrgmhyrx, mr qsppmw hym ypxvmgiw. ziwxmfypq iym
wqsh hspsv psviq, ix eggyqwer evgy mrxivhyq typzmrev.

Decrypt: maecenas lacinia eu lorem quis vulputate. nam congue, ante nec tempor rutrum, ex
lectus vestibulum felis, sit amet feugiat quam mauris quis dui. fusce nulla justo, mattis
vel augue sed, ornare dapibus nisi. etiam consectetur, elit vel bibendum tincidunt, mi ex
blandit urna, at posuere nisi mi at purus. proin eu erat magna. aliquam ac lectus nulla. c
ras pretium orci dolor. pellentesque quis odio augue. sed dignissim vestibulum ullamcorper
. vestibulum facilisis ligula eget dolor tincidunt, in mollis dui ultrices. vestibulum eui
smod dolor lorem, et accumsan arcu interdum pulvinar.

```

Рисунок 2 – Шифр Цезаря

```

1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: 2

Enter key A and B (number): 21 5

Encrypt: xflvlstf cfvrsrf lj cnylx djrt ejcijofol. sfx vnsbjl, fsol slv olxiny yjoyjx, lu
clvojt eltorajcx glcrt, tro fxlo gljbrfo djfx xfjyrt djrt qjr. gjtvl sjccf mjtton, xfoort
elc fjbjl tlq, nysfyl qfirajt srtr. lorfx vnstlvoloy, lcro elc aralsqjx orsvrqjso, xr lu
acfsqro jysf, fo intjlyl srtr xr fo iyyjt. iynrs lj lyfo xfbsf. fcrdjfx fv clvojt sjccf. v
yft iylorjx nyvr qncny. ilcclsoltdjl djrt nqrn fjbjl. tlq qrsrttrx eltorajcx jccfxvnyily
. eltorajcx gfvrctrtrt crbjcf lblo qncny orsvrqjso, rs xncrt qjr jcoyrvlt. eltorajcx ljr
txnq qncny cnylx, lo fvvjxtfs fyvj rsolyqjx ijcersfy.

Decrypt: maecenas lacinia eu lorem quis vulputate. nam congue, ante nec tempor rutrum, ex
lectus vestibulum felis, sit amet feugiat quam mauris quis dui. fusce nulla justo, mattis
vel augue sed, ornare dapibus nisi. etiam consectetur, elit vel bibendum tincidunt, mi ex
blandit urna, at posuere nisi mi at purus. proin eu erat magna. aliquam ac lectus nulla. c
ras pretium orci dolor. pellentesque quis odio augue. sed dignissim vestibulum ullamcorper
. vestibulum facilisis ligula eget dolor tincidunt, in mollis dui ultrices. vestibulum eui
smod dolor lorem, et accumsan arcu interdum pulvinar.

```

Рисунок 3 – Аффинный шифр

```

1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: 3

Enter key (text): skey

Encrypting text: ekiawxegdkggfsecmvspwusacsdszyrsdiyfkqagxkskwellorcudikhyvpmdivsekivdogrm
czckdmzmvykxopgkkgwglkqclpisyserieekypacusachsakjskmilmvpybewrgkqyldmqnopymqyckohygbryjoh
yhsfksxmqaikirakqagxwcdirmbecdsxtwvfgtorbmwxgfmmbmxxyesivtvelvsxsjxeysdtmkeipwxmqawmylzyym
cenjymlweipsdqyxyeysvmomqyuvialewlmvpysmvykzvcslsykbgbgvypmjktcdvillowomousacsbayesyeiyykoh
baqrgkcmknowrallyjmwjyjdkqagbtcjkzckdmzmvykxkggdswwgvmmemvecyoxbgvslpsraanyllkmleypjachsaeprj
sgckkzckdmzmvykwemqeyhbgvspdyvcekirsmgsecelsbgsaxxcjnykheptaxeps

Key length: 4

Decrypt key: skey

Encrypting text: maecenaslaciniaeuoremqisvulputateanamconguaeanteneconnectempporrutrumaexlectu
svestibulumfelisasitametfeugiatquamaurisquisduiafusenullajustoamattisvelauguesedaornared
apibusnisiaetiamconsecteturaelitvelbibendumtinciduntamiexblanditurnaaposuerenisimiatpuru
saproineueratmagnaaliquamalectusnullaacraspretiumorcidolorapellentesquequisodioaugueased
dignissimvestibulumnullamcorperavestibulumfacilisisligulaegetdolortinciduntainmollisduiltri
icesavestibulumuismodolorloremaetaccumsanarcuinterdumapulvinara

```

Рисунок 4 – Шифр Виженера

```

1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: 4

Enter key (text): skey

Encrypt: ekiawxeqsveaaxmysoyydyvcekusacetmvtslkxcskryekgmfqycskelloelwmerwmtmjkslbykskivs
vialewynowralyjmwedwvmqskwglkekwdedwekgsdeomkqyekpaceomsywyvemspyquoelmpystyqlyeyekxrace
twveymqycscibskspfkvcnalyqsmqakeclseksmslkogrdypski jadetwvezalilveqylsraanyllkekakivs
lpyfmrsevliskeylktmkeipwkrgekakerszypmceyhbsgfkissovylkqyyxeyskpgieekskydogrmcelmvpyskg
pscenjoxgmwemjmmyvypmjkenwvpcfdiqieiymqsyhggkesyeiyscibsnmefswqawetwcxgtepsekyjdkqagbtcj
ketwcxgtepsekjyuspgkswydsksdkecyoxyvypmjkgfmmbmxyssryeypjacebm sesddvguowysfiqlsfsdeqywem
qeyhyvypmjkmjqoysoxysmgsecelskvamkmllovbmwenmvzgfkvys

Decrypt: maecenasalaciniaaeualoremaquisavulputateaanamaconguaeaaanteanecatemporarutrumaaxa
lectusavestibulumafelisaasitaametafeugiataquamamaurisaquisaduaafusceanullaajustoamattisa
velaagueasedaornareadapibusanisiaetiamaconsecteturaaelitavelabibendumatinciduntamiaeaxa
blanditaurnaaaataposuereanisiaaatapurusaaproinaeuaeratamagnaaaaliquamaacalectusanullaac
rasapretiumaorciadoloraapellentesqueaquisaodioaaugueaasedadignissimavestibulumauillamcorper
aavestibulumafacilisisaligulaegetadoloratinciduntaaanamollisaduiaultricesaavestibulumaeui
smodadoloraloremaetaaccumsanaarcuainterdu mapulvinaraa

```

Рисунок 4 – Метод гаммирования

```

1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: 5

Enter key (number): 5

Encrypting: `mbeoeral baoui`ie !uoler!mtqriv muupatet .an!mncgndu ,oaetn bet leap rurrtlu ,
yel beut!sevusculuf meri ,hs tlauef tehguag `u!m`msuriq hulsud/if rudcn mualj ruot ,`mtt
riv mea fudus ee ,soanerd qacirun ri/ie itlac oodsucuesu ,meuiv meb cioeud!mitcneiou,tm !i
yeb aldnuui nr-aa toptsseleinhsm !iuap suru .rpho ntee ar t`mog/aa iltqlaa !celucrun mual
.scrap eritluo cr!iodol.rp meeltnretq!etqrio id!otatg/es eed fiinrsliv reitubul!mmualbmso
ep.rv reitubul!mafhcilhs!siltgale dg todol ritcneiou,ti nmllrid huu tlirdc/sv reitubul!m
terinm dodol roler-me tbatcrmoaa cr!uoietdr lup muivan.r

Decrypting: maecenas lacinia eu lorem quis vulputate. nam congue, ante nec tempor rutrum,
ex lectus vestibulum felis, sit amet feugiat quam mauris quis dui. fusce nulla justo, matt
is vel augue sed, ornare dapibus nisi. etiam consectetur, elit vel bibendum tincidunt, mi
ex blandit urna, at posuere nisi mi at purus. proin eu erat magna. aliquam ac lectus nulla
. cras pretium orci dolor. pellentesque quis odio augue. sed dignissim vestibulum ullamcor
per. vestibulum facilisis ligula eget dolor tincidunt, in mollis dui ultrices. vestibulum
euismod dolor lorem, et accumsan arcu interdum pulvinar.

```

Рисунок 5 – Сети Фейстеля

```

1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5. Feisel; 6. RSA
Select cipher method: 6
N open: 589
E open: 209
D secret: 509
Crypt number: 30
Decrypt number: 216
Rand number: 216
Success!

```

Рисунок 6 – шифр RSA

### **3. Вывод**

В ходе домашней работы были изучены различные методы шифрования, такие как шифр Цезаря, Аффинного шифра, шифр Виженера, метод гаммирования, сети Фейстеля, RSA. Программа реализована на объектно-ориентированном языке программирования C++ с проектной сборкой CMake. Было реализовано шифрование и расшифрование сообщений. Исходные данные алфавита и сообщений были считаны с файла.



## ПРИЛОЖЕНИЕ

Листинг 1

```
#include <iostream>
#include <string>
#include <ctime>
#include "file_manager.h"
#include "caesar.h"
#include "affine.h"
#include "vegenere.h"
#include "xorcipher.h"
#include "feistel.h"
#include "rsa.h"

int main() {
    srand(time(NULL));
    std::string text;
    std::string alphabet;

    FileManager fileManager;
    text = fileManager.reader(text, "res\\text.txt");
    alphabet = fileManager.reader(alphabet, "res\\alphabet.txt");

    int n_case;
    std::cout << "1. Caesar; 2. Affine; 3. Vegenere, 4. XOR cipher; 5.
Feistel; 6. RSA" << std::endl;
    std::cout << "Select cipher method: ";
    std::cin >> n_case;
    if (n_case == 1) {
        int key;
        std::cout << "\nEnter key (number): ";
        std::cin >> key;
        Caesar caesar(key, alphabet, text);
    } else if (n_case == 2) {
        int key_a, key_b;
        std::cout << "\nEnter key A and B (number): ";
        std::cin >> key_a >> key_b;
        Affine affine(key_a, key_b, alphabet, text);
    } else if (n_case == 3) {
        std::string key_string;
        std::cout << "\nEnter key (text): ";
        std::cin >> key_string;
        Vegenere vegenere(key_string, text, alphabet);
    } else if (n_case == 4) {
        std::string key_string;
        std::cout << "\nEnter key (text): ";
        std::cin >> key_string;
        XORCipher xorCipher(key_string, text, alphabet);
    } else if (n_case == 5) {
        std::string key_string;
        std::cout << "\nEnter key (number): ";
        std::cin >> key_string;
        Feistel Feistel(key_string, text);
    } else if (n_case == 6) {
        RSA rsa;
    }
}
```

```

#include <iostream>
#include <fstream>
#include "service_manager.h"
#include "file_manager.h"

std::string FileManager::reader(std::string input, std::string path) {
    ServiceManager serviceManager;
    std::ifstream in_input(path);
    if (in_input.is_open()) std::getline(in_input, input);
    in_input.close();
    return serviceManager.setLowerLetter(input);
}

FileManager::FileManager() {}

```

```

#include <algorithm>
#include "service_manager.h"

std::string ServiceManager::setLowerLetter(std::string input) {
    std::transform(input.begin(), input.end(), input.begin(),
        [](unsigned char c) { return std::tolower(c); });
    return input;
}

std::string ServiceManager::removeSpace(std::string input) {
    input.erase(std::remove_if(input.begin(), input.end(),
        isspace), input.end());
    return input;
}

std::string ServiceManager::shiftText(std::string input, std::string
    alphabet, int shift) {
    for(auto & word : input){
        for (unsigned int i = 0; i < alphabet.size(); i++) {
            if (word == alphabet.at(i)) {
                word = alphabet.at((i + shift) % alphabet.size());
                break;
            }
        }
    }
    return input;
}

int ServiceManager::searchNumInput(int num, std::string input, std::string
    input_search) {
    for (unsigned int i = 0; i < input_search.size(); i++) {
        if (input[num] == input_search[i]) {
            return i;
        }
    }
    return 0;
}

ServiceManager::ServiceManager() {}

```

```

#include <iostream>
#include "caesar.h"

void Caesar::encrypting() {
    int size = alphabet_.size();
    for(auto & word : words_){
        for (int i = 0; i < size; i++) {
            if (word == alphabet_.at(i)) {
                word = alphabet_.at((i + key_) % size);
                break;
            }
        }
    }
}

void Caesar::decrypting() {
    int size = alphabet_.size();
    for(auto & word : words_){
        for (int i = 0; i < size; i++) {
            if (word == alphabet_.at(i)) {
                if (i >= key_) {
                    word = alphabet_.at((i - key_) % size);
                } else {
                    word = alphabet_.at((i + size - key_) % size);
                }
                break;
            }
        }
    }
}

Caesar::Caesar(int key, std::string &alphabet, std::string &words) :
key_(key), alphabet_(alphabet), words_(words) {
    encrypting();
    std::cout << "\nEncrypt: " << words <<std::endl;

    decrypting();
    std::cout << "\nDecrypt: " << words <<std::endl;
}

```

```

#include <iostream>
#include "affine.h"

void Affine::encrypting() {
    if (nod(key_a_, alphabet_.size()) == 1) {
        int size = alphabet_.size();
        for (auto &word : words_) {
            for (int i = 0; i < size; i++) {
                if (word == alphabet_.at(i)) {
                    word = alphabet_.at((key_a_ * i + key_b_) % size);
                    break;
                }
            }
        }
    } else {
        std::cout << "Key and not simple!" << std::endl;
    }
}

```

```

}

void Affine::decrypting() {
    int size = alphabet_.size();
    int a_inverse = Foo(key_a_, alphabet_.size());
    if (nod(key_a_, alphabet_.size()) == 1) {
        for (auto &word : words_) {
            for (int i = 0; i < size; i++) {
                if (word == alphabet_.at(i)) {
                    if (i >= key_b_) {
                        word = alphabet_.at(a_inverse * (i - key_b_) % size);
                    } else {
                        word = alphabet_.at(a_inverse * (i + size - key_b_) %
size);
                    }
                    break;
                }
            }
        }
    }
}

int Affine::nod(int key_a, int alphabet_size) {
    return alphabet_size ? nod(alphabet_size, key_a % alphabet_size) : key_a;
}

int Affine::Foo(int key_a, int alphabet_size) {
    int x, y;
    int gcd = GCD(key_a, alphabet_size, x, y);
    if (gcd != 1) return 1;
    return (x % alphabet_size + alphabet_size) % alphabet_size;
}

int Affine::GCD(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    int x1, y1;
    int d = GCD(b % a, a, x1, y1);
    x = y1 - (b/a) * x1;
    y = x1;
    return d;
}

Affine::Affine(int keyA, int keyB, std::string &alphabet, std::string &words)
: key_a_(keyA), key_b_(keyB), alphabet_(alphabet), words_(words) {
    encrypting();
    std::cout << "\nEncrypt: " << words <<std::endl;

    decrypting();
    std::cout << "\nDecrypt: " << words <<std::endl;
}

```

```

#include <iostream>
#include <vector>
#include <algorithm>
#include "vegenere.h"
#include "service_manager.h"

void Vegenere::encrypting() {
    ServiceManager serviceManager;
    text_ = serviceManager.removeSpace(text_);
    int j = 0;
    for (unsigned int i = 0; i < text_.size(); i++) {
        text_[i] = alphabet_[(serviceManager.searchNumInput(i, text_,
alphabet_ + serviceManager.searchNumInput(j, key_, alphabet_)) %
alphabet_.size())];
        j++;
        if (j >= key_.size()) j = 0;
    }
    std::cout << "\nEncrypting text: " << text_ << std::endl;
}

void Vegenere::decrypting() {
    ServiceManager serviceManager;
    std::vector<double> FREQ_ENGLISH = {0.0816, 0.0149, 0.0202, 0.0425,
0.1270, 0.0222, 0.0201, 0.0609, 0.0696, 0.0015,
0.0129, 0.0402, 0.0240, 0.0674,
0.0750, 0.0192, 0.0009, 0.0598, 0.0632, 0.0935,
0.0275, 0.0095, 0.0256, 0.0015,
0.0199, 0.0007};
    int key_length = 2;
    // поиск длины ключа
    for (key_length; key_length < 12; key_length++) {
        if (indexOfCoincidence(key_length, text_) > IOC) break;
    }
    std::cout << "\nKey length: " << key_length << std::endl;

    // поиск букв ключа
    for (int i = 0; i < key_length; i++) {
        std::string text_shift;
        std::vector<double> buffer_diff;
        // разбиваем на группы
        for (unsigned int j = i; j < text_.length(); j += key_length) {
            text_shift.push_back(text_[j]);
        }
        for (unsigned int n = 0; n < alphabet_.size(); n++) {
            // частота группы
            std::vector<double> freqs;
            for (auto &letter : alphabet_) {
                freqs.push_back(std::count(text_shift.begin(),
text_shift.end(), letter) * 1.0 / text_shift.size());
            }
            // сравнение с частотами
            double diff = 0;
            for (unsigned int m = 0; m < alphabet_.size(); m++) {
                diff += abs(freqs[m] - FREQ_ENGLISH[m]);
            }
            text_shift = serviceManager.shiftText(text_shift, alphabet_, 1);
            buffer_diff.push_back(diff);
        }
    }
}

```

```

        int index = std::min_element(buffer_diff.begin(), buffer_diff.end())
- buffer_diff.begin();
        key_dec_.push_back(alphabet_[alphabet_.size() - index]);
    }
    std::cout << "\nDecrypt key: " << key_dec_ << std::endl;
    int a = 0;
    for (unsigned int i = 0; i < text_.size(); i++) {
        text_[i] = alphabet_[(serviceManager.searchNumInput(i, text_,
alphabet_) - serviceManager.searchNumInput(a, key_dec_, alphabet_)
+ alphabet_.size()) % alphabet_.size()];
        a++;
        if (a >= key_.size()) a = 0;
    }
    std::cout << "\nEncrypting text: " << text_ << std::endl;
}

// индекс совпадений
double Vegenere::indexOfCoincidence(int shift, std::string input) {
    std::string buffer_text;
    // создаем буффер со сдвигом
    for (unsigned int i = 0; i < input.size(); i++) {
        buffer_text.push_back(input[i]);
        i += shift - 1;
    }
    // подсчет индекса совпадений
    double ioc = 0;
    for (auto &letter : alphabet_) {
        auto n = std::count(buffer_text.begin(), buffer_text.end(), letter);
        ioc += static_cast<double>(n * (n - 1)) /
static_cast<double>(buffer_text.size() * (buffer_text.size() - 1));
    }
    return ioc;
}

Vegenere::Vegenere(std::string &key, std::string &text, std::string
&alphabet) : key_(key), text_(text), alphabet_(alphabet) {
    encrypting();
    decrypting();
}

```

Листинг 7

```

#include <iostream>
#include "xorcipher.h"
#include "service_manager.h"

void XORCipher::encrypting() {
    // сложение по модулю N
    ServiceManager serviceManager;
    int j = 0;
    for (unsigned int i = 0; i < text_.length(); i++) {
        text_[i] = alphabet_[(serviceManager.searchNumInput(i, text_,
alphabet_) +
                                serviceManager.searchNumInput(j, key_,
alphabet_))
                                % alphabet_.length()];
        j++;
        if (j >= key_.size()) j = 0;
    }
}

void XORCipher::decrypting() {
    ServiceManager serviceManager;

```

```

        int j = 0;
        for (unsigned int i = 0; i < text_.length(); i++) {
            text_[i] = alphabet_[(serviceManager.searchNumInput(i, text_,
alphabet_) + alphabet_.length()
                                - serviceManager.searchNumInput(j, key_,
alphabet_))
                                % alphabet_.length()];
            j++;
            if (j >= key_.size()) j = 0;
        }
    }
    XORCipher::XORCipher(std::string &key, std::string &text, std::string
&alphabet) : key_(key), text_(text), alphabet_(alphabet) {
        encrypting();
        std::cout << "\nEncrypt: ";
        std::cout << text << std::endl;

        decrypting();
        std::cout << "\nDecrypt: ";
        std::cout << text << std::endl;
    }
}

```

Листинг 8

```

#include <iostream>
#include "feistel.h"

void Feistel::encrypting() {
    char temp = '0', l = '0', r = '0';
    for (unsigned int i = 0; i < text_.size(); i += 2)
    {
        l = text_[i];
        r = text_[i + 1];
        for (int i = 0; i < key_.size(); i++)
        {
            temp = r ^ fGamma(l, key_, i);
            r = l;
            l = temp;
        }
        text_[i] = l;
        text_[i + 1] = r;
    }
}

void Feistel::decrypting() {
    char temp = '0', l = '0', r = '0';
    for (unsigned int i = 0; i < text_.size(); i += 2) // рассматриваем блоки
по 2 символа
    {
        l = text_[i];
        r = text_[i + 1];
        for (int i = key_.size() - 1; i >= 0; i--) // n раундов
        {
            temp = l ^ fGamma(r, key_, i);
            l = r;
            r = temp;
        }
        text_[i] = l;
        text_[i + 1] = r;
    }
}

char Feistel::fGamma(char subblock, std::string key, int round) {

```

```

        return (subblock * key[round]) % static_cast<char>(pow(2, round) + 1);
    }
    Feistel::Feistel(std::string &key, std::string &text) : key_(key),
    text_(text) {
        encrypting();
        std::cout << "\nEncrypting: ";
        std::cout << text << std::endl;

        decrypting();
        std::cout << "\nDecrypting: ";
        std::cout << text << std::endl;
    }

```

Листинг 9

```

#include <iostream>
#include "rsa.h"

void RSA::generateKey() {
    int q;
    int p = getRandPrime();
    while (1) {
        q = getRandPrime();
        if (p != q) break;
    }

    // ОТКРЫТЫЙ КЛЮЧ - n
    n = p * q;
    int f = (p - 1) * (q - 1);

    e = getRandEPrime(f);

    // СЕКРЕТНЫЙ КЛЮЧ - d
    d = ModInverse(e, f);

    std::cout << "N open: " << n << std::endl;
    std::cout << "E open: " << e << std::endl;
    std::cout << "D secret: " << d << std::endl;
}

void RSA::encrypting() {
    // генерируем случайное число - исходный текст
    number = rand() % n;
    crypt_number = ModPow(number, e, n);
    std::cout << "Crypt number: " << crypt_number << std::endl;
}

void RSA::decrypting() {
    int decrypt_number = ModPow(crypt_number, d, n);
    std::cout << "Decrypt number: " << decrypt_number << std::endl;
    std::cout << "Rand number: " << number << std::endl;
    if (number == decrypt_number) {
        std::cout << "Success!" << std::endl;
    } else {
        std::cout << "Fail!" << std::endl;
    }
}

bool RSA::isPrime(int number) {
    for (int i = 2; i <= sqrt(number); i++) {
        if (number % i == 0) return false;
    }
    return true;
}

```



```

}
int RSA::isEPrime(int number, int f) {
    int a = f;
    int b = number;
    while (b > 0) {
        int q = a / b;
        int r = a % b;
        a = b;
        b = r;
    }
    int gcd = a;
    return a == 1;
}

int RSA::getRandPrime() {
    while (true) {
        int number = rand() % 100 + 10;
        if (isPrime(number)) return number;
    }
}

int RSA::getRandEPrime(int f) {
    while(1) {
        int number = rand() % f;
        if (isEPrime(number, f)) return number;
    }
}

int RSA::ModInverse(int e, int modulus) {
    for (int d = 1; d <= 0x7FFFFFFF; d++) {
        long prod = (e * d) % modulus;
        if (prod == 1) return d;
    }
    return 0;
}

int RSA::ModPow(int val, int exponent, int modulus) {
    long result = 1;
    for (int i=1; i <= exponent; i++) result = (result*val) % modulus;
    return (int)result;
}

RSA::RSA() {
    generateKey();
    encrypting();
    decrypting();
}

```