Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра «Информационных технологий и систем»

**Лабораторная работа №2**   
по дисциплине:

«Защита информации»

Разработал:

Студент группы 8091

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_» \_\_\_\_\_2021г

Проверил:

Жгун Т. В. \_\_\_\_\_\_

«\_\_» \_\_\_\_\_2021г

**Великий Новгород**

**2021**

**Задача:**

Зашифровать фрагмент текста объемом около 500 символов (допустим диапазон 500-1000 символов). Использовать следующие способы шифрования: ECB, CBC, OFB, CFB.   
Построить гистограммы для полученных зашифрованных текстов. Проанализировать результат (четыре гистограммы).

**Описание алгоритмов:**

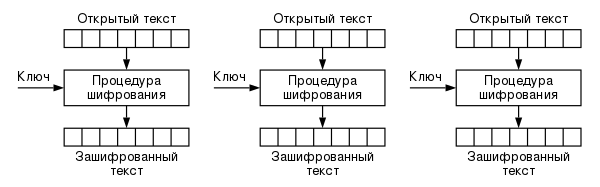
**Electronic Codebook(ECB)**

В ГОСТ 28147—89 этот режим называется **режимом простой замены**.

*Шифрование:*

Пусть дано сообщение **{\displaystyle P}P** (открытый текст, последовательность бит, данные).

Во время шифрования выполняются следующие действия:

[](https://commons.wikimedia.org/wiki/File:ECB_Encryption_ru.svg?uselang=ru)

Шифрование в режиме ECB (режиме электронной кодовой книги)

1. Сообщение делится на блоки одинакового размера. Размер (длина) блока равен ***n*** и измеряется в битах. В результате получается последовательность блоков {\displaystyle P\_{1},P\_{2},...,P\_{q}}**P1, P2, P3, … Pq** Последний блок при необходимости дополняется до длины ***n***../.
2. Каждый блок {\displaystyle P\_{i}}**Pi** шифруется алгоритмом шифрования {\displaystyle E\_{k}}**Ek**с использованием ключа *k*:

 {\displaystyle C\_{i}=E\_{k}\left(P\_{i},k\right),}

где:

* *i* — номер блока;
* *k* — [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F));
* {\displaystyle P\_{i}}*Pi*— блок сообщения (открытый текст);
* {\displaystyle C\_{i}}*Ci* — зашифрованный блок (шифротекст);
* {\displaystyle E\_{k}}*Ek* — функция, выполняющая блочное шифрование.

В результате получаются зашифрованные блоки **{\displaystyle C\_{1},C\_{2},...,C\_{q}}C1, C2, C3, … Cq**.

*Расшифровка:*

выполняется функцией {\displaystyle D\_{k}}Dk с использованием того же ключа *k*:

 {\displaystyle P\_{i}=D\_{k}\left(C\_{i},k\right).}

Особенности:

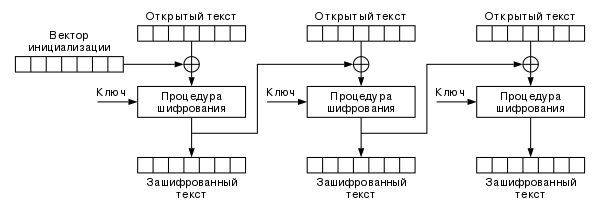
* каждый блок шифруется/расшифровывается независимо от других блоков.

Недостатки ECB:

* сохранение статистических особенностей [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) (поскольку одинаковым блокам [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст) соответствуют одинаковые блоки [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82)).

### Cipher Block Chaining (CBC)

Для шифрования некоторого сообщения {\displaystyle P}**P** выполняются следующие действия.

[](https://commons.wikimedia.org/wiki/File:CBC_Encryption_ru.svg?uselang=ru)

Шифрование в режиме CBC (режиме сцепления блоков шифротекста)

* Сообщение разбивается на блоки одинакового размера. Размер (длина) блока равен ***n*** и измеряется в битах. При необходимости последний блок дополняется до длины {\displaystyle n}***n***../.
* Шифрование очередного (*i*-го) блока сообщения ({\displaystyle P\_{i}}**Pi**) выполняется с использованием предыдущего зашифрованного (*(i-1)*-го) блока ({\displaystyle C\_{i-1}}**Сi-1**). Для первого блока ({\displaystyle P\_{1}}**Pi**) зашифрованного блока ({\displaystyle C\_{0}}**Ci-1**) не существует, поэтому первый блок шифруют с использованием «вектора инициализации»

{\displaystyle C\_{0}=IV}  (вектор инициализации — случайное число)

Размер (длина) IV равна размеру блока (*n*).

* В функцию шифрования {\displaystyle E\_{k}}**Ek** передаётся [сумма по модулю 2 («⊕», «xor»)](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) текущего блока сообщения {\displaystyle P\_{i}}**Pi**и предыдущего зашифрованного блока {\displaystyle C\_{i-1}}**Ci-1**:

{\displaystyle C\_{i}=E\_{k}\left(P\_{i}\oplus C\_{i-1},k\right),} 

где:

* *i* — номер блока;
* *k* — [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F));
* *IV* — вектор инициализации ([синхропосылка](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BF%D0%BE%D1%81%D1%8B%D0%BB%D0%BA%D0%B0&action=edit&redlink=1" \o "Синхропосылка (страница отсутствует)));
* {\displaystyle P\_{i}} *Pi* — блок сообщения ([открытый текст](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82));
* {\displaystyle C\_{i-1}}*Сi-1* — зашифрованный блок ([шифротекст](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст)), полученный на предыдущем шаге шифрования;
* {\displaystyle E\_{k}}*Ek* — функция, выполняющая [блочное шифрование](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Расшифровка выполняется функцией **{\displaystyle D\_{k}}Dk** с использованием тех же ключа *k* и вектора инициализации *IV*:

{\displaystyle C\_{0}=IV}{\displaystyle P\_{i}=C\_{i-1}\oplus D\_{k}\left(C\_{i},k\right)}

Недостатки CBC:

* возможность определения начала изменения данных по изменению [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст) (если сравнить шифротексты двух сообщений с одним и тем же ключом, то номер первого блока, в котором шифротексты различаются, будет соответствовать номеру первого блока, в котором различаются исходные сообщения);
* возможность изменения [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) при перемещении блоков
* возможность изменения блока [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст) {\displaystyle C\_{i-1}} **Сi-1** путём изменения блока [сообщения](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) **Pi** {\displaystyle P\_{i}};
* невозможность [распараллеливания](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC) шифрования (поскольку для шифрования каждого *i*-го блока требуется блок, зашифрованный на предыдущем шаге (блоки связаны между собой))[[1]](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F#cite_note-six-1).

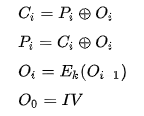
Достоинства CBC:

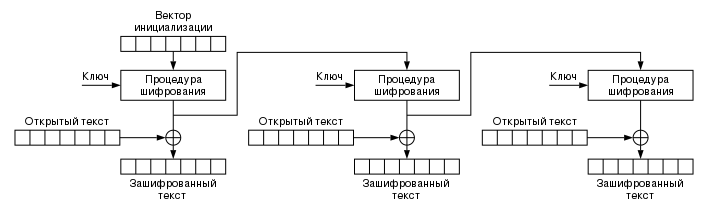
* постоянная скорость обработки блоков (скорость определяется эффективностью реализации шифра; время выполнения операции «[xor](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2" \o "Сложение по модулю 2)» пренебрежимо мало);
* отсутствие статистических особенностей, характерных для режима ECB (поскольку каждый блок [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) «смешивается» с блоком [шифротекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o "Шифротекст), полученным на предыдущем шаге шифрования);
* возможность [распараллеливания](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC) расшифровки[[1]](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F#cite_note-six-1).

### Output Feedback (OFB)

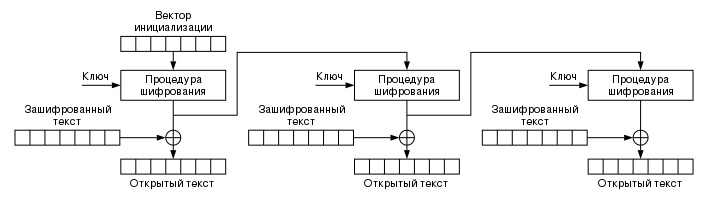
Режим (OFB) обратной связи вывода превращает блочный шифр в синхронный шифр потока: он генерирует ключевые блоки, которые являются результатом сложения с блоками открытого текста, чтобы получить зашифрованный текст. Так же, как с другими шифрами потока, зеркальное отражение в зашифрованном тексте производит зеркально отражённый бит в открытом тексте в том же самом местоположении. Это свойство позволяет многим кодам с исправлением ошибок функционировать как обычно, даже когда исправление ошибок применено перед кодированием.

Из-за симметрии операции сложения, шифрование и расшифрование похожи:

 {\displaystyle C\_{i}=P\_{i}\oplus O\_{i}}{\displaystyle P\_{i}=C\_{i}\oplus O\_{i}}{\displaystyle O\_{i}=E\_{k}(O\_{i-1})}{\displaystyle O\_{0}=IV}

[](https://commons.wikimedia.org/wiki/File:OFB_Encryption_ru.svg?uselang=ru)

Шифрование в режиме **OFB**

[](https://commons.wikimedia.org/wiki/File:OFB_Decryption_ru.svg?uselang=ru)

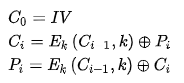
Расшифрование в режиме **OFB**

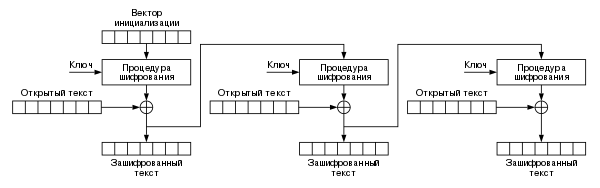
Каждая операция блочного шифра обратной связи вывода зависит от всех предыдущих и поэтому не может быть выполнена параллельно. Однако, из-за того, что открытый текст или зашифрованный текст используются только для конечного сложения, операции блочного шифра могут быть выполнены заранее, позволяя выполнить заключительное шифрование параллельно с открытым текстом.  
  
Обратная связь по выходу на k разрядов не рекомендуется из соображений криптостойкости. Режим OFB имеет следующее преимущество по сравнению с режимом CFB: ошибки, возникающие в результате передачи по каналу с шумом, при дешифровании не «размазываются» по всему шифротексту, а локализуются в пределах одного блока. Однако открытый текст может быть изменён путём определённых манипуляций с блоками шифротекста. Несмотря на то, что OFB-шифрование не поддаётся распараллеливанию, эффективность процедуры может быть повышена за счёт предварительной генерации независимой последовательности блоков.

Данный метод называется также «режим обратной связи по выходу».

### Cipher Feedback (CFB)

Во время шифрования каждый блок открытого текста складывается по модулю 2 с блоком, зашифрованным на предыдущем шаге.



[](https://commons.wikimedia.org/wiki/File:CFB_Encryption_ru.svg?uselang=ru)

Шифрование в режиме обратной связи по шифротексту

Криптостойкость CFB определяется криптостойкостью используемого шифра. Блоки открытого текста «смешиваются» («маскируются») с блоками шифротекста. Если в режиме CFB с полноблочной обратной связью имеется два идентичных блока шифротекста, результат, например, шифрования алгоритмом DES на следующем шаге будет тем же. Скорость шифрования режима CFB с полноблочной обратной связью та же, что и у блочного шифра, причём возможности распараллеливания процедуры шифрования ограничены.

**Реализация**

Поставленная задача решена с помощью языка программирования С++ на фреймворке QT.

**Описание интерфейса**

Программа предоставляет пользовательское окно, которое поделено на две вкладки:

- Работа с текстом

- Отображение гистограмм

Пользователь во вкладке работы с текстом имеет возможность ввода следующих данных:

- Текст, необходимый для анализа

- K0

- K1

- P0

- C0

Во вкладке отображение гистограмм, пользователь может увидеть работу алгоритмов шифрования в графическом виде.

Ось x – буква в 10 виде (а=0, б=1, в=2 и тд)

Ось y – количество букв.

**Код**

Код представлен только в виде основного выполняемого окна, остальные файлы, такие как (Заголовки, визуализация окна), не будут присутствовать в отчёте по функциональной ненадобности.

Полный код и исходные файлы (в том числе исполняемый .exe файл) можно найти в репозитории по ссылке:

https://github.com/VasilevIvanVladimirovich/Information-Security-Course/tree/main/Lab2

Запуск алгоритмов работы программы осуществляется по нажатию на кнопку “Расчёт”.

Файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "qcustomplot.h"

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

: QMainWindow(*parent*), ui(new Ui::MainWindow) {

ui->setupUi(this);

}

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

delete ui;

}

void MainWindow::**bin**(int n, QVector<int> &bit\_array) //функция перевода числа в двоичную систему счисления

{

int arr[5];

for (int i = 4; i >= 0; i--) {

arr[i] = n % 2;

n = n / 2;

}

for (int i = 0; i < 5; i++) {

bit\_array.push\_back(arr[i]);

}

}

double MainWindow::**MaxValue**(QVector<int> &gen, int n) {

double max = gen[0];

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

if (gen[i] > max) max = gen[i];

}

return max;

}

double MainWindow::**MinValue**(QVector<int> &gen, int n) {

double min = gen[0];

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

if (gen[i] < min) min = gen[i];

}

return min;

}

void MainWindow::**Word**(QString str, QVector<int> &bit\_array) // функция перевода строки в двоичный код

{

int \*arr\_num = new int[str.length()];

QString smal[32] = {"а", "б", "в", "г", "д", "е", "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"};

QString big[32] = {"А", "Б", "В", "Г", "Д", "Е", "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"};

for (int i = 0; i < str.length(); i++) //перебор символов и запись их в вектор двоичных чисел

{

for (int j = 0; j < 32; j++) {

if (str[i] == smal[j]) {

arr\_num[i] = j;

bin(arr\_num[i], *bit\_array*);

break;

} else if (str[i] == big[j]) {

arr\_num[i] = j;

bin(arr\_num[i], *bit\_array*);

break;

}

}

}

delete[] arr\_num;

}

void MainWindow::**invertLastBit**(QVector<int> &out\_bit) {

int back;

if (out\_bit.back() == 0) back = 1;

else if (out\_bit.back() == 1) back = 0;

out\_bit.pop\_back();

out\_bit.push\_back(back);

}

void MainWindow::**sum**(QVector<int> &out\_bit, int \*buf\_array, int \*K) {

for (int i = 0; i < 5; i++) {

if ((buf\_array[i] == 1 && K[i] == 1) || (buf\_array[i] == 0 && K[i] == 0)) out\_bit.push\_back(0);

if ((buf\_array[i] == 1 && K[i] == 0) || (buf\_array[i] == 0 && K[i] == 1)) out\_bit.push\_back(1);

}

}

void MainWindow::**ECB**(QVector<int> &bit\_array, QVector<int> &ECB\_out\_bit, int \*K0, int \*K1) {

int buf\_array[5];

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++) {

buf\_array[j] = bit\_array[j + i];

}

if (buf\_array[4] == 0) sum(*ECB\_out\_bit*, *buf\_array*, *K0*); //P1+K

else sum(*ECB\_out\_bit*, *buf\_array*, *K1*);

invertLastBit(*ECB\_out\_bit*);

}

}

void **convertBinareDecimal**(QVector<int> &out\_bit, QVector<int> &ECB\_out) {

int sum;

for (int i = 0; i < out\_bit.length(); i += 5) {

sum = 0;

for (int j = 4, k = 0; j >= 0; j--, k++) {

if (out\_bit[i + k] == 1) sum += pow(2, j);

}

ECB\_out.push\_back(sum);

}

}

void MainWindow::**CPC**(QVector<int> &bit\_array, QVector<int> &CPC\_out\_bit, int \*C0) {

int C[5];

for (int i = 0; i < 5; i++) C[i] = C0[i];

int buf[5];

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++) {

buf[j] = bit\_array[i + j];

}

sum(*CPC\_out\_bit*, *buf*, *C*); //P1+C0

if (CPC\_out\_bit[i] == 0) CPC\_out\_bit[i] = 1;

else if (CPC\_out\_bit[i] == 1) CPC\_out\_bit[i] = 0;

for (int k = 0; k < 5; k++) C[k] = CPC\_out\_bit[k + i];

}

}

void MainWindow::**sumArray**(int \*arr1, int \*arr2, int \*result) {

for (int i = 0; i < 5; i++) {

if ((arr1[i] == 1 && arr2[i] == 1) || (arr1[i] == 0 && arr2[i] == 0)) result[i] = 0;

if ((arr1[i] == 1 && arr2[i] == 0) || (arr1[i] == 0 && arr2[i] == 1)) result[i] = 1;

}

}

void MainWindow::**OFB**(QVector<int> &bit\_array, QVector<int> &OFB\_out\_bit, int \*C0, int \*P0) {

int buf\_sum[5];

int bin\_word[5];

int C[5];

int P[5];

for (int i = 0; i < 5; i++) C[i] = C0[i];

for (int i = 0; i < 5; i++) P[i] = P0[i];

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++) bin\_word[j] = bit\_array[i + j];

sumArray(*C*, *P*, *buf\_sum*); //C0+P0

sumArray(*bin\_word*, *buf\_sum*, *C*); //P1+(C0+P0)

if (C[4] == 0) C[4] = 1;

else if (C[4] == 1) C[4] = 0;

for (int j = 0; j < 5; j++) P[j] = bin\_word[j];

for (int j = 0; j < 5; j++) OFB\_out\_bit.push\_back(C[j]);

}

}

void MainWindow::**CFB**(QVector<int> &bit\_array, QVector<int> &OFB\_out\_bit, int \*C0) {

int C[5];

int buf[5];

for (int i = 0; i < 5; i++) C[i] = C0[i];

if (C[0] == 0) C[0] = 1;

else if (C[0] == 1) C[0] = 0;

for (int i = 0; i < bit\_array.length(); i += 5) {

for (int j = 0; j < 5; j++) buf[j] = bit\_array[i + j];

sum(*OFB\_out\_bit*, *buf*, *C*); //P1+С0

invertLastBit(*OFB\_out\_bit*);

for (int j = 0; j < 5; j++) C[j] = OFB\_out\_bit[i + j];

}

}

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

ui->OFB->clearPlottables(); //очистка гистограмм

ui->CPC->clearPlottables();

ui->CFB->clearPlottables();

ui->ECB->clearPlottables();

ui->textPlot->clearPlottables();

QString text = ui->textEdit->toPlainText(); //ввод с окна

QString k0 = ui->lineEdit\_4->text();

QString k1 = ui->lineEdit\_3->text();

QString p0 = ui->lineEdit->text();

QString c0 = ui->lineEdit\_2->text();

int K0[5];

int K1[5];

int P0[5];

int C0[5];

////////////////////////////////////////Перевод входных значений в int

if (k0 == "" && k1 == "" && p0 == "" && c0 == "") {

k0 = "10001";

k1 = "00001";

p0 = "00000";

c0 = "11100";

ui->lineEdit\_4->setText(k0);

ui->lineEdit\_3->setText(k1);

ui->lineEdit->setText(p0);

ui->lineEdit\_2->setText(c0);

}

if (k0.length() == 5) {

for (int i = 0; i < 5; i++) {

if (k0[i] == '0') K0[i] = 0;

else if (k0[i] == '1') K0[i] = 1;

else ui->lineEdit\_4->setText("Ошибка ввода");

}

} else ui->lineEdit\_4->setText("Ошибка ввода");

if (k1.length() == 5) {

for (int i = 0; i < 5; i++) {

if (k1[i] == '0') K1[i] = 0;

else if (k1[i] == '1') K1[i] = 1;

else ui->lineEdit\_3->setText("Ошибка ввода");

}

} else ui->lineEdit\_3->setText("Ошибка ввода");

if (p0.length() == 5) {

for (int i = 0; i < 5; i++) {

if (p0[i] == '0') P0[i] = 0;

else if (p0[i] == '1') P0[i] = 1;

else ui->lineEdit->setText("Ошибка ввода");

}

} else ui->lineEdit->setText("Ошибка ввода");

if (c0.length() == 5) {

for (int i = 0; i < 5; i++) {

if (c0[i] == '0') C0[i] = 0;

else if (c0[i] == '1') C0[i] = 1;

else ui->lineEdit\_2->setText("Ошибка ввода");

}

} else ui->lineEdit\_2->setText("Ошибка ввода");

////////////////////////////////////////////////////////////Чтение и перевод в двоичный код текста

QVector<int> bit\_array;

QVector<int> text\_decimal;

Word(text, *bit\_array*);

convertBinareDecimal(*bit\_array*, *text\_decimal*);

QString out = "";

////////////////////////////////////////////////////////////График ECB

QVector<int> ECB\_out\_bit;

QVector<int> ECB\_out;

ECB(*bit\_array*, *ECB\_out\_bit*, *K0*, *K1*);

convertBinareDecimal(*ECB\_out\_bit*, *ECB\_out*);

int N = ECB\_out.length();

double h, hh = 0;

double ni;

int m = 32;

double a, b;

a = MinValue(*ECB\_out*, N);

b = MaxValue(*ECB\_out*, N);

h = 1;

QVector<double> x1(m), y1(m);

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

ni = 0;

for (int j = 0; j < N; j++) {

if (ECB\_out[j] >= double(a) + hh and ECB\_out[j] < double(a) + hh + h) ni++;

else continue;

}

x1[i] = double(a) + hh + (h / 2);

y1[i] = ni;

hh += h;

}

double first\_maxY = 1;

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

if (y1[i] > first\_maxY) first\_maxY = y1[i];

}

ui->ECB->xAxis->setRange(0, 32);

ui->ECB->yAxis->setRange(0, first\_maxY);

QCPBars \*bars1 = new QCPBars(*ui->ECB->xAxis*, *ui->ECB->yAxis*);

bars1->setData(x1, y1, true);

bars1->setWidth(h);

ui->ECB->replot();

///////////////////////////////////////////// График CPC

QVector<int> CPC\_out\_bit;

QVector<int> CPC\_out;

CPC(*bit\_array*, *CPC\_out\_bit*, *C0*);

convertBinareDecimal(*CPC\_out\_bit*, *CPC\_out*);

N = CPC\_out.length();

hh = 0;

a = MinValue(*CPC\_out*, N);

b = MaxValue(*CPC\_out*, N);

h = 1;

QVector<double> x2(m), y2(m);

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

ni = 0;

for (int j = 0; j < N; j++) {

if (CPC\_out[j] >= double(a) + hh and CPC\_out[j] < double(a) + hh + h) ni++;

else continue;

}

x2[i] = double(a) + hh + (h / 2);

y2[i] = ni;

hh += h;

}

ui->CPC->xAxis->setRange(0, 32);

ui->CPC->yAxis->setRange(0, first\_maxY); //сюда надо вставить максимальное h

QCPBars \*bars2 = new QCPBars(*ui->CPC->xAxis*, *ui->CPC->yAxis*);

bars2->setData(x2, y2, true);

bars2->setWidth(h);

ui->CPC->replot();

////////////////////////////////////////////////////////////График OFB

QVector<int> OFB\_out\_bit;

QVector<int> OFB\_out;

OFB(*bit\_array*, *OFB\_out\_bit*, *C0*, *P0*);

convertBinareDecimal(*OFB\_out\_bit*, *OFB\_out*);

N = OFB\_out.length();

hh = 0;

a = MinValue(*OFB\_out*, N);

b = MaxValue(*OFB\_out*, N);

h = 1;

QVector<double> x3(m), y3(m);

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

ni = 0;

for (int j = 0; j < N; j++) {

if (OFB\_out[j] >= double(a) + hh and OFB\_out[j] < double(a) + hh + h) ni++;

else continue;

}

x3[i] = double(a) + hh + (h / 2);

y3[i] = ni;

hh += h;

}

ui->OFB->xAxis->setRange(0, 32);

ui->OFB->yAxis->setRange(0, first\_maxY);

QCPBars \*bars3 = new QCPBars(*ui->OFB->xAxis*, *ui->OFB->yAxis*);

bars3->setData(x3, y3, true);

bars3->setWidth(h);

ui->OFB->replot();

////////////////////////////////////////////////////////////График CFB

QVector<int> CFB\_out\_bit;

QVector<int> CFB\_out;

CFB(*bit\_array*, *CFB\_out\_bit*, *C0*);

convertBinareDecimal(*CFB\_out\_bit*, *CFB\_out*);

N = CFB\_out.length();

hh = 0;

a = MinValue(*CFB\_out*, N);

b = MaxValue(*CFB\_out*, N);

h = 1;

QVector<double> x4(m), y4(m);

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

ni = 0;

for (int j = 0; j < N; j++) {

if (CFB\_out[j] >= double(a) + hh and CFB\_out[j] < double(a) + hh + h) ni++;

else continue;

}

x4[i] = double(a) + hh + (h / 2);

y4[i] = ni;

hh += h;

}

ui->CFB->xAxis->setRange(0, 32);

ui->CFB->yAxis->setRange(0, first\_maxY);

QCPBars \*bars4 = new QCPBars(*ui->CFB->xAxis*, *ui->CFB->yAxis*);

bars4->setData(x4, y4, true);

bars4->setWidth(h);

ui->CFB->replot();

//////////////////////////////////////////////////////////// График Текста

N = text\_decimal.length();

hh = 0;

a = MinValue(*text\_decimal*, N);

b = MaxValue(*text\_decimal*, N);

h = 1;

QVector<double> x0(m), y0(m);

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

ni = 0;

for (int j = 0; j < N; j++) {

if (text\_decimal[j] >= double(a) + hh and text\_decimal[j] < double(a) + hh + h) ni++;

else continue;

}

x0[i] = double(a) + hh + (h / 2);

y0[i] = ni;

hh += h;

}

ui->textPlot->xAxis->setRange(0, 32);

ui->textPlot->yAxis->setRange(0, first\_maxY);

QCPBars \*bars0 = new QCPBars(*ui->textPlot->xAxis*, *ui->textPlot->yAxis*);

bars0->setData(x0, y0, true);

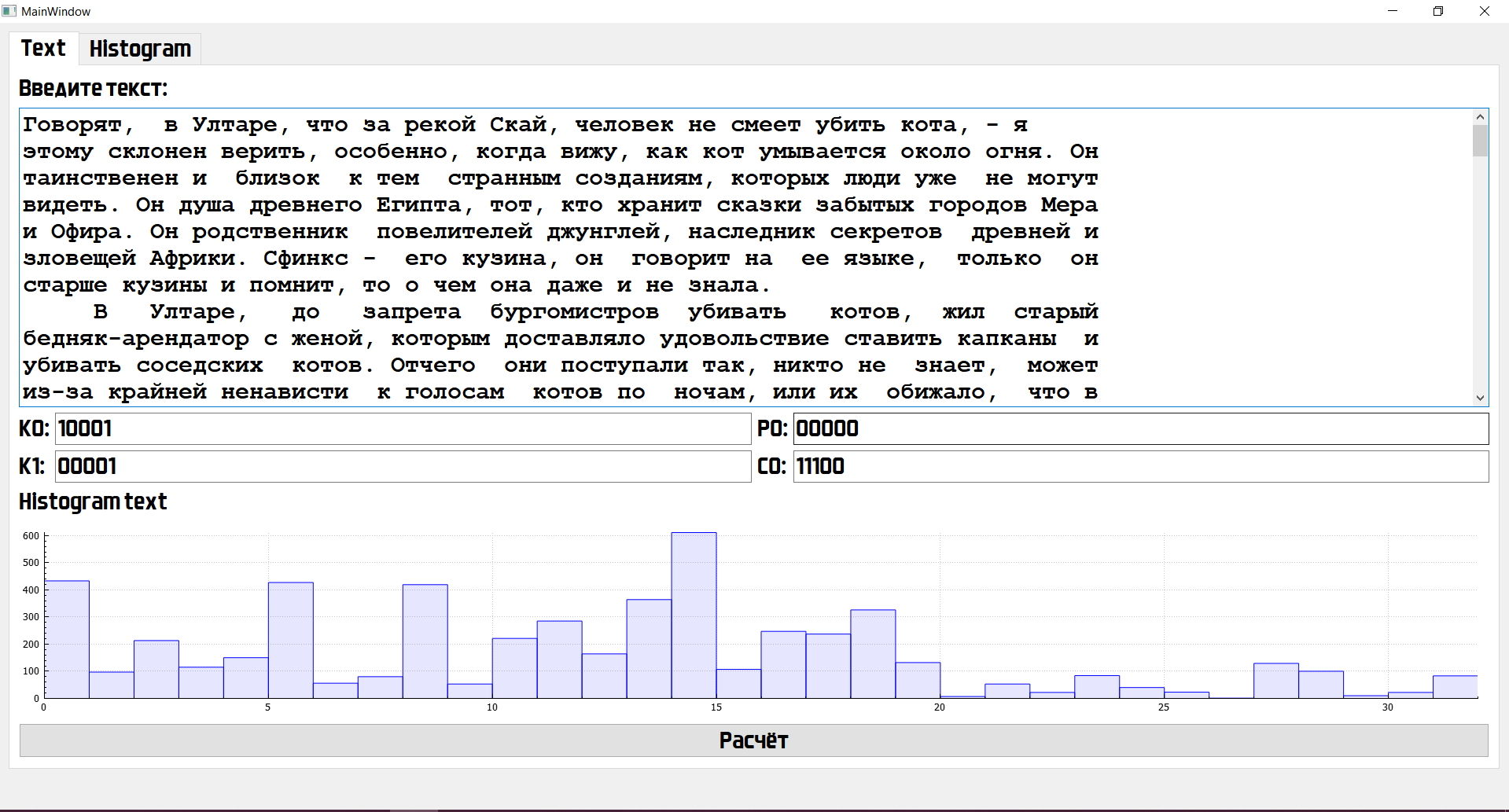
bars0->setWidth(h);

ui->textPlot->replot();

}

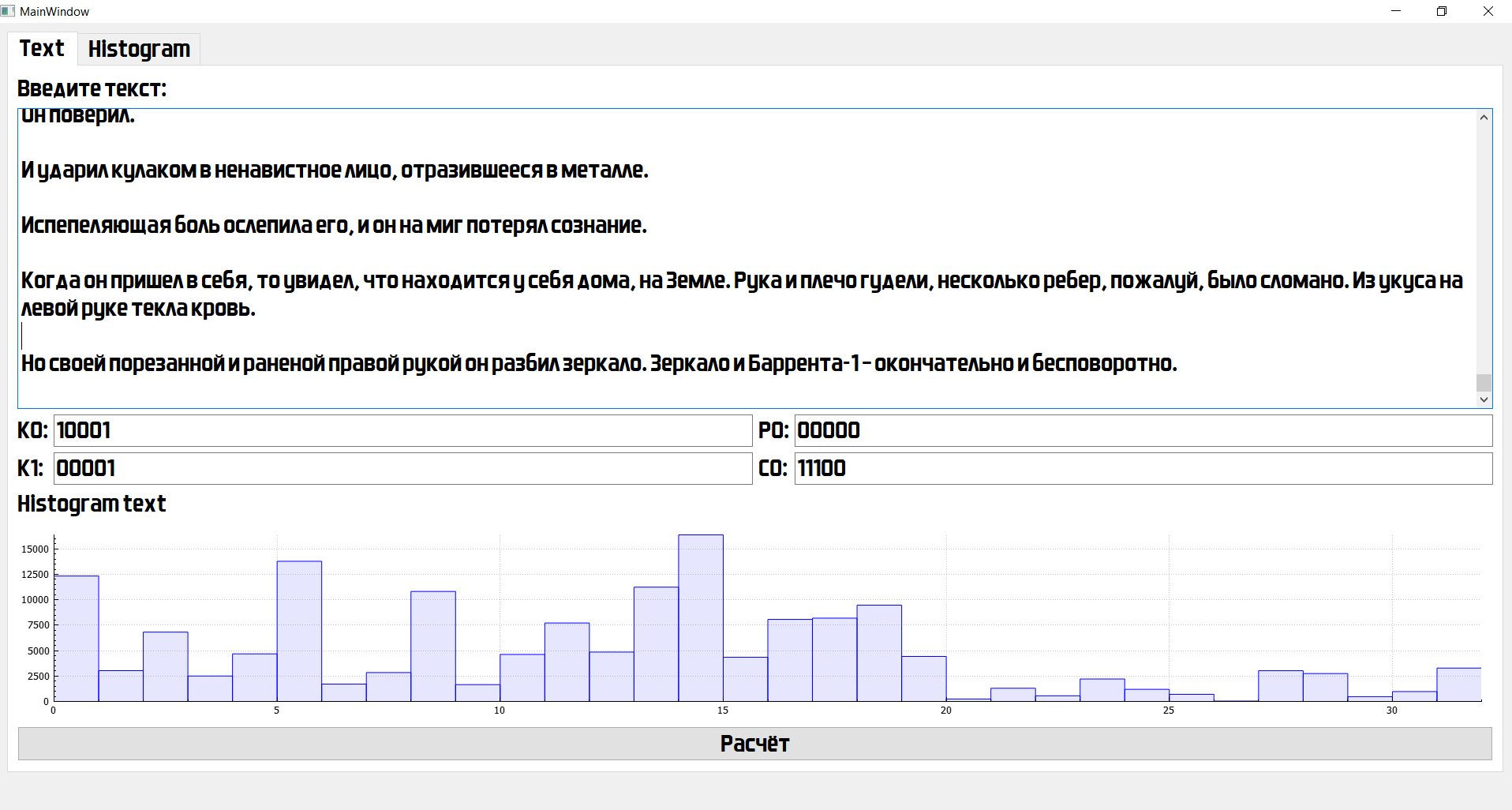
**Пример работы программы**

## Для первого примера, я взял произведение Говарда Ф. Лавкрафта “Коты Ултара” объёмом в 1013 слов.





## Для второго примера, я взял произведение Роберта Шекли “Цивилизация статуса” объёмом в 27693 слов.





**Вывод:**

В результате визуального анализа двух примеров, можно утверждать, что CPC и СFB шифрование лучше всего подходят, так как их распределение ближе всего совпадает с равномерным, следовательно враг будет видеть только шум.