Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Прикладная математика и компьютерная безопасность

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 16**

по Криптографические методы защиты информации

наименование дисциплины

Программная реализация шифра DES

тема

Преподаватель В.И. Вайнштейн

подпись**,** дата инициалы, фамилия

Студент КИ17-01, 031722011 К.А. Василенко

номер группы, зачетной книжки подпись**,** дата инициалы, фамилия

Красноярск 2021

# ВВЕДЕНИЕ

Цель работы: реализовать шифр DES (шифрование и расшифрование) на любом языке программирования. Предусмотреть графический интерфейс.

Задачи:

1. реализовать программно шифр DES, предусматривая графический интерфейс;
2. провести тесты на работоспособность программы;
3. сделать отчёт о проделанной работе.

**Реализация шифра**

1. **Описание шифра**

**DES** (англ. *Data Encryption Standard*) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований.

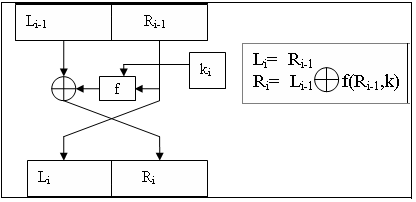


Рисунок 1 - Прямое преобразование сетью Фейстеля.

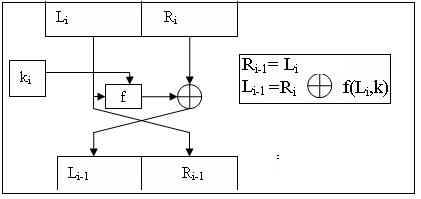


Рисунок 2 - Обратное преобразование сетью Фейстеля.

**Зашифрование:**

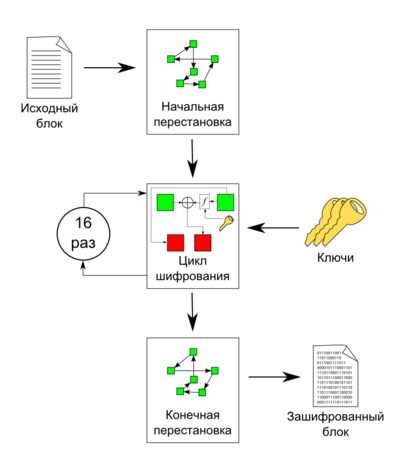


Рисунок 3 - Схема шифрования алгоритма DES.

Исходный текст — блок 64 бит. Процесс шифрования состоит из начальной перестановки, 16 циклов шифрования и конечной перестановки.

**Начальная перестановка**

Исходный текст T {\displaystyle T} T (блок 64 бит) преобразуется c помощью начальной перестановки I P {\displaystyle \mathrm {IP} } IP, которая определяется таблицей 1:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. Начальная перестановка IP | | | | | | | | | | | | | | | |
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

**Циклы шифрования**

Полученный после начальной перестановки 64-битовый блок IP(T) участвует в 16 циклах преобразования Фейстеля.

— 16 циклов преобразования Фейстеля:

Разбить IP(T) на две части L 0 , R 0 {\displaystyle L\_{0},R\_{0}} L0, R0, где L 0 , R 0 {\displaystyle L\_{0},R\_{0}} L0, R0 — соответственно 32 старших битов и 32 младших битов блока T 0 {\displaystyle T\_{0}} T0 IP(T)= L0R0

L 0 R 0 {\displaystyle L\_{0}R\_{0}} LL

Пусть T i − 1 = L i − 1 R i − 1 {\displaystyle T\_{i-1}=L\_{i-1}R\_{i-1}} Ti-1 = Li-1Ri-1 результат (i-1) итерации, тогда результат i-ой итерации Ti = LiRi T i = L i R i {\displaystyle T\_{i}=L\_{i}R\_{i}} определяется:

Li = Ri-1

Ri = Li-1 xor f(Ri-1, ki)

L i = R i − 1 {\displaystyle L\_{i}=R\_{i-1}} R i = L i − 1 ⊕ f ( R i − 1 , k i ) {\displaystyle R\_{i}=L\_{i-1}\oplus f(R\_{i-1},k\_{i})} Левая половина LiL i {\displaystyle L\_{i}} равна правой половине предыдущего вектора L i − 1 R i − 1 {\displaystyle L\_{i-1}R\_{i-1}} Li-1Ri-1. А правая половина R i {\displaystyle R\_{i}} Ri — это битовое сложение Li-1 L i − 1 {\displaystyle L\_{i-1}} и f(Ri-1, ki) f ( R i − 1 , k i ) {\displaystyle f(R\_{i-1},k\_{i})} по модулю 2.

В 16-циклах преобразования Фейстеля функция f играет роль шифрования.

**Основная функция шифрования (функция Фейстеля)**

Аргументами функции f {\displaystyle f} f являются 32-битовый вектор Ri-1R i − 1 {\displaystyle R\_{i-1}} и 48-битовый ключ k i {\displaystyle k\_{i}} ki, который является результатом преобразования 56-битового исходного ключа шифра k {\displaystyle k} k. Для вычисления функции f {\displaystyle f} f последовательно используются:

1. функция расширения E {\displaystyle \mathrm {E} } E,
2. сложение по модулю 2 с ключом k i {\displaystyle k\_{i}} ki
3. преобразование S {\displaystyle \mathrm {S} } S, состоящее из 8 преобразований S {\displaystyle \mathrm {S} } S-блоков S 1 , S 2 , S 3 …   S 8 {\displaystyle \mathrm {S} \_{1},\mathrm {S} \_{2},\mathrm {S} \_{3}\ldots \ \mathrm {S} \_{8}}
4. перестановка P {\displaystyle \mathrm {P} } P.

Функция E {\displaystyle \mathrm {E} } E расширяет 32-битовый вектор Ri-1R i − 1 {\displaystyle R\_{i-1}} до 48-битового вектора E ( R i − 1 ) {\displaystyle \mathrm {E} (R\_{i-1})} путём дублирования некоторых битов из R i − 1 {\displaystyle R\_{i-1}} Ri-1; порядок битов вектора E ( R i − 1 ) {\displaystyle \mathrm {E} (R\_{i-1})} указан в таблице 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 2. Функция расширения E | | | | | |
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

R i − 1 {\displaystyle R\_{i-1}} Полученный после перестановки блок E ( R i − 1 ) {\displaystyle \mathrm {E} (R\_{i-1})} E(Ri-1) складывается по модулю 2 с ключами k i {\displaystyle k\_{i}} ki и затем представляется в виде восьми последовательных блоков B 1 , B 2 , . . . B 8 {\displaystyle B\_{1},B\_{2},...B\_{8}} B1, … , B8.

E(Ri-1) xor ki = B1B2 … B8

E ( R i − 1 ) ⊕ k i = B 1 B 2 . . . B 8 {\displaystyle \mathrm {E} (R\_{i-1})\oplus k\_{i}=B\_{1}B\_{2}...B\_{8}} Каждый B j {\displaystyle B\_{j}} Bj является 6-битовым блоком. Далее каждый из блоков B j {\displaystyle B\_{j}} Bj трансформируется в 4-битовый блок Bj’B j ′ {\displaystyle B'\_{j}} с помощью преобразований S j {\displaystyle S\_{j}} Sj. Преобразования S j {\displaystyle S\_{j}} Sj определяются таблицей 3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3. Преобразования S i {\displaystyle S\_{i}} Si, i=1…8 | | | | | | | | | | | | | | | | |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 1 | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 2 | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 3 | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 1 | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 2 | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 3 | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 1 | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 2 | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 2 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 3 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 2 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 3 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 1 | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 2 | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 3 | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 2 | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 3 | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

**Генерирование ключей ki**

Ключи k i {\displaystyle k\_{i}} ki получаются из начального ключа k {\displaystyle k} k (56 бит = 7 байтов или 7 символов в ASCII) следующим образом. Добавляются биты в позиции 8, 16, 24, 32, 40, 48, 56, 64 ключа kk {\displaystyle k} таким образом, чтобы каждый байт содержал нечетное число единиц. Это используется для обнаружения ошибок при обмене и хранении ключей. Затем делают перестановку для расширенного ключа (кроме добавляемых битов 8, 16, 24, 32, 40, 48, 56, 64). Такая перестановка определена в таблице 5. **k i {\displaystyle k\_{i}}**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 5. | | | | | | | | | | | | | |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 58 | 50 | 42 | 34 | 26 | 18 |
| 10 | 2 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | 60 | 52 | 44 | 36 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 | 62 | 54 | 46 | 38 | 30 | 22 |
| 14 | 6 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | 28 | 20 | 12 | 4 |

Эта перестановка определяется двумя блоками C 0 {\displaystyle C\_{0}} C0 и D0D 0 {\displaystyle D\_{0}} по 28 бит каждый. C i , D i {\displaystyle C\_{i},D\_{i}} CiDi i=1,2,3…получаются из Ci-1Di-1C i − 1 , D i − 1 {\displaystyle C\_{i-1},D\_{i-1}} одним или двумя левыми циклическими сдвигами согласно таблице 6.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 6. | | | | | | | | | | | | | | | | |
| **i** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| Число сдвига | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Ключ k i {\displaystyle k\_{i}} ki, i=1,…16 состоит из 48 бит, выбранных из битов вектора C i D i {\displaystyle C\_{i}D\_{i}} CiDi (56 бит) согласно таблице 7. Первый и второй биты k i {\displaystyle k\_{i}} ki есть биты 14, 17 вектора CiDiC i D i {\displaystyle C\_{i}D\_{i}}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 7. | | | | | | | | | | | | | | | |
| 14 | 17 | 11 | 24 | 1 | 5 | 3 | 28 | 15 | 6 | 21 | 10 | 23 | 19 | 12 | 4 |
| 26 | 8 | 16 | 7 | 27 | 20 | 13 | 2 | 41 | 52 | 31 | 37 | 47 | 55 | 30 | 40 |
| 51 | 45 | 33 | 48 | 44 | 49 | 39 | 56 | 34 | 53 | 46 | 42 | 50 | 36 | 29 | 32 |

**Конечная перестановка**

Конечная перестановка IP-1I P − 1 {\displaystyle \mathrm {IP} ^{-1}} действует на T 16 − 1 {\displaystyle T\_{16}^{-1}} T16-1 (где T 16 − 1 = R 16 + L 16 {\displaystyle T\_{16}^{-1}=R\_{16}+L\_{16}} T16-1 = L16 + R16) и является обратной к первоначальной перестановке. Конечная перестановка определяется таблицей 8.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 8. Обратная перестановка I P − 1 {\displaystyle \mathrm {IP} ^{-1}} | | | | | | | | | | | | | | | |
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 | 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 | 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 | 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 | 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

**Расшифрование:**

При расшифровании данных все действия выполняются в обратном порядке. В 16 циклах расшифрования, в отличие от шифрования c помощью прямого преобразования сетью Фейстеля, здесь используется обратное преобразование сетью Фейстеля.

Ri-1 = Li

Li-1 = Ri xor f(Li, ki)

R i − 1 = L i {\displaystyle R\_{i-1}=L\_{i}} L i − 1 = R i ⊕ f ( L i , k i ) {\displaystyle L\_{i-1}=R\_{i}\oplus f(L\_{i},k\_{i})} Ключ k i {\displaystyle k\_{i}} ki, i=16,…,1, функция f, перестановка IP и I P − 1 {\displaystyle IP^{-1}} IP-1 такие же, как и в процессе шифрования. Алгоритм генерации ключей зависит только от ключа пользователя, поэтому при расшифровании они идентичны.

**Режимы использования DES**

1. Режим электронной кодовой книги (ECB — Electronic Codebook): обычное использование DES как блочного шифра. Шифруемый текст разбивается на блоки, при этом каждый блок шифруется отдельно, не взаимодействуя с другими блоками.

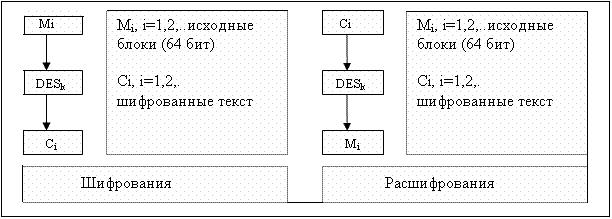


Рисунок 4 – ECB.

1. Режим сцепления блоков шифротекста (СВС — Cipher Block Chaining). Каждый очередной блок M i {\displaystyle M\_{i}} Mi i>=1, перед зашифровыванием складывается по модулю 2 с предыдущим блоком зашифрованного текста C i − 1 {\displaystyle C\_{i-1}} Ci-1. Вектор C 0 {\displaystyle C\_{0}} C0 — начальный вектор, он меняется ежедневно и хранится в секрете.

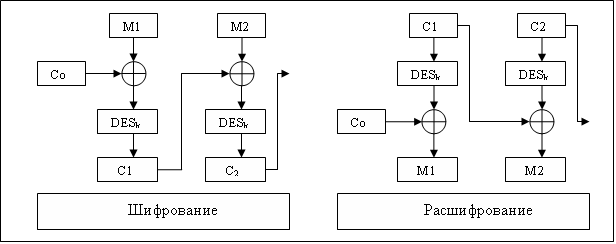


Рисунок 5 – CBC.

1. Режим обратной связи по шифротексту (Cipher Feedback). В режиме CFB вырабатывается блочная «гамма» Z 0 , Z 1 , . . . {\displaystyle Z\_{0},Z\_{1},...} Z i = D E S k ( C i − 1 ) {\displaystyle Z\_{i}=DES\_{k}(C\_{i-1})} Z0, Z1, …, Zi = DESk(Ci-1)C i = M i ⊕ Z i {\displaystyle C\_{i}=M\_{i}\oplus Z\_{i}} Ci = Mi xor Zi. Начальный вектор C 0 {\displaystyle C\_{0}} C0 является синхропосылкой и предназначен для того, чтобы разные наборы данных шифровались по-разному с использованием одного и того же секретного ключа. Синхропосылка посылается получателю в открытом виде вместе с зашифрованным файлом. Алгоритм DES, в отличие от предыдущих режимов, используется только как шифрование (в обоих случаях).

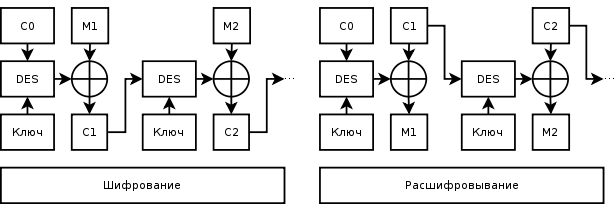


Рисунок 6 – CFB.

1. Режим обратной связи по выходу (OFB — Output Feedback). В режиме OFB вырабатывается блочная «гамма» Z 0 , Z 1 , . . . {\displaystyle Z\_{0},Z\_{1},...} Z i = D E S k ( Z i − 1 ) C i = M i ⊕ Z i {\displaystyle Z\_{i}=DES\_{k}(Z\_{i-1})C\_{i}=M\_{i}\oplus Z\_{i}} Z0, Z1, …, Zi = DESk(Zi-1)C i = M i ⊕ Z i {\displaystyle C\_{i}=M\_{i}\oplus Z\_{i}} Ci = Mi xor Zi, i>=1. Режим также использует DES только как шифрование (в обоих случаях).

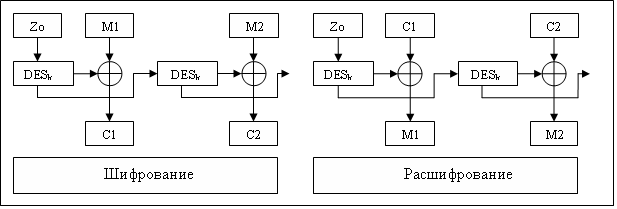


Рисунок 7 – OFB.

{\displaystyle c\_{j}=(m\_{j}+k\_{j})\mod {n}}{\displaystyle m\_{j}=(c\_{j}+n-k\_{j})\mod {n}}**Программная реализация**

Программа написана на языке Python с использованием библиотеки PyQt5 и Qt designer для отрисовки графического интерфейса.

1. **Листинг с описанием основного алгоритма программы**

**des.py**

**from** PyQt5 **import** QtWidgets  
**from** tkinter **import** Tk, Label  
**import** time  
  
**def** sdvig(str: str, n: int):  
 arr = []  
 **for** i **in** range(len(str)):  
 arr.append('')  
 **for** i **in** range(len(arr)):  
 arr[(i - n) % len(arr)] = str[i]  
 arrTxt = ''  
 **for** i **in** arr:  
 arrTxt += i  
 **return** arrTxt  
  
  
**def** xor(a: str, b: str):  
 **if** len(a) > len(b):  
 a = a.zfill(len(b))  
 **elif** len(b) > len(a):  
 b = b.zfill(len(a))  
 res = ""  
 **for** i **in** range(len(a)):  
 **if** a[i] == '1':  
 **if** b[i] == '1':  
 res += '0'  
 **else**:  
 res += '1'  
 **if** a[i] == '0':  
 **if** b[i] == '1':  
 res += '1'  
 **else**:  
 res += '0'  
 **return** res  
  
  
**def** extreme\_bits(blocks6b: str):  
 b = blocks6b[:1] + blocks6b[5:]  
 **return** int(b, 2)  
  
  
**def** middle\_bits(blocks6b: str):  
 b = blocks6b[1:5]  
 **return** int(b, 2)  
  
  
**def** substitutions(block48b: str):  
 \_\_Sbox = [  
 [ # 0  
 [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],  
 [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],  
 [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],  
 [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],  
 ],  
 [ # 1  
 [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],  
 [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],  
 [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],  
 [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9],  
 ],  
 [ # 2  
 [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],  
 [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],  
 [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],  
 [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],  
 ],  
 [ # 3  
 [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],  
 [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],  
 [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],  
 [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],  
 ],  
 [ # 4  
 [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],  
 [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],  
 [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],  
 [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],  
 ],  
 [ # 5  
 [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],  
 [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],  
 [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],  
 [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],  
 ],  
 [ # 6  
 [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],  
 [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],  
 [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],  
 [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],  
 ],  
 [ # 7  
 [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],  
 [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],  
 [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],  
 [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],  
 ]  
 ]  
 blocks6b = []  
 **for** i **in** range(8):  
 blocks6b.append(block48b[:6])  
 block48b = block48b[6:]  
 blocks32b = ''  
 **for** i **in** range(len(blocks6b)):  
 two\_b = extreme\_bits(blocks6b[i])  
 four\_b = middle\_bits(blocks6b[i])  
 blocks32b+=bin(\_\_Sbox[i][two\_b][four\_b])[2:].zfill(4)  
 **return** blocks32b  
  
  
**def** func\_F(R0: str, key: str):  
 \_\_EP = [  
 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,  
 8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,  
 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,  
 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1,  
 ]  
 \_\_P = [  
 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,  
 2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25,  
 ]  
 expansion\_permutation = initial\_permutation(R0, \_\_EP, 48)  
 block48b = xor(expansion\_permutation, key)  
 block32b = substitutions(block48b)  
 **return** initial\_permutation(block32b, \_\_P, 32)  
  
  
**def** round\_feistel\_scheme(L0: str, R0: str, key: str):  
 tmp = R0  
 R0 = func\_F(R0, key)  
 R0 = xor(R0, L0)  
 L0 = tmp  
 **return** L0, R0  
  
**def** feistel\_scheme(L0: str, R0: str, keys, flag: int):  
 **if** flag == 1:  
 **for** i **in** range(len(keys)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, keys[i])  
 L0, R0 = R0, L0  
 **else**:  
 **for** i **in** range(len(keys)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, keys[len(keys) - 1 - i])  
 L0, R0 = R0, L0  
 **return** L0, R0  
  
  
**def** genKeys(key8b: str):  
 \_\_CP = [  
 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,  
 23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,  
 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,  
 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32,  
 ]  
 \_\_K1P = [  
 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,  
 10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,  
 ]  
 \_\_K2P = [  
 63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,  
 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4,  
 ]  
 keys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 key8bBin = ''  
 **for** i **in** key8b:  
 key8bBin += bin(ord(i))[2:].zfill(8)  
  
 N1 = initial\_permutation(key8bBin, \_\_K1P, 28)  
 N2 = initial\_permutation(key8bBin, \_\_K2P, 28)  
 **for** i **in** range(16):  
 **if** i == 0 **or** i == 1 **or** i == 8 **or** i == 15:  
 n = 1  
 **else**:  
 n = 2  
 N = sdvig(N1, n) + sdvig(N2, n)  
 keys[i] = initial\_permutation(N, \_\_CP, 48)  
 **return** keys  
  
  
# начальная перестановка  
# строка в дв СС и по какому массиву переставляем  
**def** initial\_permutation(binStr: str, arr: list, length: int):  
 afterPermutation = []  
 **for** i **in** range(length):  
 afterPermutation.append('')  
 **for** i **in** range(len(afterPermutation)):  
 afterPermutation[i] = binStr[arr[i] - 1]  
 afterPermutationStr = ''  
 **for** i **in** afterPermutation:  
 afterPermutationStr += i  
 **return** afterPermutationStr  
  
  
**def** codingDES(block: str, flag: int, keys: list):  
 \_\_IP = [  
 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,  
 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,  
 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,  
 61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7,  
 ]  
 \_\_FP = [  
 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,  
 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,  
 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,  
 34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25,  
 ]  
  
 afterInitPerm = initial\_permutation(block, \_\_IP, 64)  
  
 L0 = afterInitPerm[:32]  
 R0 = afterInitPerm[32:]  
 L16, R16 = feistel\_scheme(L0, R0, keys, flag)  
 block64b = initial\_permutation(L16 + R16, \_\_FP, 64)  
 **return** block64b  
  
**def** ECB(keys1, text, flag: int):  
 start = time.time()  
 **if not** keys1 **or** keys1 == '':  
 keys = genKeys('DESkey56')  
 **else**:  
 **if** len(keys1) != 16 \* 48:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** keys1:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 keys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 **for** i **in** range(16):  
 keys[i] = keys1[:48]  
 keys1 = keys1[48:]  
  
 **if not** text:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Введите текст!")  
 msgBox.exec\_()  
 **return** ("")  
  
 blocksBin, file = before\_coding(text, flag)  
  
 percent = 100 / len(blocksBin)  
 progress = 0  
 normal = 0  
 root = Tk()  
  
 label = Label(text='Progressbar')  
 x = (root.winfo\_screenwidth() - root.winfo\_reqwidth()) / 2  
 y = (root.winfo\_screenheight() - root.winfo\_reqheight()) / 2  
 root.wm\_geometry("+%d+%d" % (x, y))  
 label.pack()  
  
 block64Int = b''  
 **for** i **in** range(len(blocksBin)):  
 block64b = codingDES(blocksBin[i], flag, keys)  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 #print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass  
 for** i **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
  
 blocksText = after\_coding(file, block64Int, flag)  
 **if** normal == 0:  
 root.destroy()  
 print('Время выполнения: ', (time.time() - start)/60)  
 **return** blocksText  
  
**def** before\_coding(text, flag: int):  
 file = 0  
 **if** type(text) == str:  
 **if** flag == 1:  
 textToInt = text.encode()  
 **else**:  
 textToInt = b''  
 **for** i **in** text:  
 textToInt+=bytes([ord(i)])  
 **else**:  
 textToInt = text  
 file = 1  
 print('OK')  
  
 print(textToInt)  
 blocks = []  
 **while** len(textToInt) > 0:  
 blocks.append(textToInt[:8])  
 textToInt = textToInt[8:]  
 count = 0  
 **if** len(blocks[len(blocks) - 1]) != 8:  
 **while** len(blocks[len(blocks) - 1]) != 8:  
 count += 1  
 blocks[len(blocks) - 1] += bytes([0])  
 blocks[len(blocks) - 1] = blocks[len(blocks) - 1][:-1] + bytes([count])  
 blocksBin = []  
 **for** i **in** range(len(blocks)):  
 binStr = ''  
 **for** j **in** blocks[i]:  
 binStr += bin(j)[2:].zfill(8)  
 blocksBin.append(binStr)  
 **return** blocksBin, file  
  
**def** after\_coding(file, block64Int, flag):  
 **if** file == 0:  
 **if** flag != 1:  
 numb = block64Int[len(block64Int) - 1]  
 print('n', numb)  
 blocksText1 = block64Int  
 j = 2  
 **for** i **in** range(numb-1):  
 **if** blocksText1[-j] == 0:  
 blocksText1 = blocksText1[:-1]  
 j = 1  
 **else**:  
 **break  
 if** len(blocksText1) < len(block64Int):  
 blocksText1 = blocksText1[:-1]  
 blocksText = ''  
 **for** i **in** blocksText1:  
 blocksText += chr(i)  
 **else**:  
 blocksText = ''  
 **for** i **in** block64Int:  
 blocksText += chr(i)  
 **else**:  
 **if** flag != 1:  
 numb = block64Int[len(block64Int) - 1]  
 print('n', numb)  
 blocksText = block64Int  
 j = 2  
 **for** i **in** range(numb-1):  
 **if** blocksText[-j] == 0:  
 blocksText = blocksText[:-1]  
 j = 1  
 **else**:  
 **break  
 if** len(blocksText) < len(block64Int):  
 blocksText = blocksText[:-1]  
 **else**:  
 blocksText = block64Int  
 **return** blocksText  
  
**def** CBC(keys1, text, flag: int, vector\_init:str):  
 **if not** vector\_init **or** vector\_init == '':  
 vector\_init = '1110011110010101110010111001010010000111001010011100101001001011'  
 **else**:  
 **if** len(vector\_init) != 64:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** vector\_init:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!!")  
 msgBox.exec\_()  
 **return** ("")  
 **if not** keys1 **or** keys1 == '':  
 keys = genKeys('DESkey56')  
 **else**:  
 **if** len(keys1) != 16 \* 48:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** keys1:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 keys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 **for** i **in** range(16):  
 keys[i] = keys1[:48]  
 keys1 = keys1[48:]  
 **if not** text:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Введите текст!")  
 msgBox.exec\_()  
 **return** ("")  
  
 blocksBin, file = before\_coding(text, flag)  
 percent = 100 / len(blocksBin)  
 normal = 0  
 root = Tk()  
  
 label = Label(text='Progressbar')  
 x = (root.winfo\_screenwidth() - root.winfo\_reqwidth()) / 2  
 y = (root.winfo\_screenheight() - root.winfo\_reqheight()) / 2  
 root.wm\_geometry("+%d+%d" % (x, y))  
 label.pack()  
  
 **if** flag == 1:  
 progress = 0  
 print('ZASH')  
 block64Int = b''  
 first\_step = xor(blocksBin[0], vector\_init)  
 block64b = codingDES(first\_step, flag, keys)  
 block64b1 = block64b  
 **for** k **in** range(8):  
 block64Int += bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **for** p **in** range(1, len(blocksBin)):  
 first\_step = xor(block64b1, blocksBin[p])  
 block64b = codingDES(first\_step, flag, keys)  
 block64b1 = block64b  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass** # if p < len(blocksBin) - 1:  
 # first\_step = xor(block64b, blocksBin[p+1])  
 **for** k **in** range(8):  
 block64Int += bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **else**:  
 progress = 1  
 print('RASSH')  
 block64Int = b''  
 first\_step = blocksBin[0]  
 block64b = xor(vector\_init, codingDES(blocksBin[0], flag, keys))  
 **for** k **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **for** p **in** range(1, len(blocksBin)):  
 block64b = xor(first\_step, codingDES(blocksBin[p], flag, keys))  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass  
 for** k **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 first\_step = blocksBin[p]  
  
 blocksText = after\_coding(file, block64Int, flag)  
 **if** normal == 0:  
 root.destroy()  
 **return** blocksText  
  
**def** CFB(keys1, text, flag: int, vector\_init:str):  
 **if not** vector\_init **or** vector\_init == '':  
 vector\_init = '1110011110010101110010111001010010000111001010011100101001001011'  
 **else**:  
 **if** len(vector\_init) != 64:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** vector\_init:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!!")  
 msgBox.exec\_()  
 **return** ("")  
 **if not** keys1 **or** keys1 == '':  
 keys = genKeys('DESkey56')  
 **else**:  
 **if** len(keys1) != 16 \* 48:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** keys1:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 keys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 **for** i **in** range(16):  
 keys[i] = keys1[:48]  
 keys1 = keys1[48:]  
 **if not** text:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Введите текст!")  
 msgBox.exec\_()  
 **return** ("")  
  
 blocksBin, file = before\_coding(text, flag)  
  
 percent = 100 / len(blocksBin)  
 progress = 1  
 normal = 0  
 root = Tk()  
  
 label = Label(text='Progressbar')  
 x = (root.winfo\_screenwidth() - root.winfo\_reqwidth()) / 2  
 y = (root.winfo\_screenheight() - root.winfo\_reqheight()) / 2  
 root.wm\_geometry("+%d+%d" % (x, y))  
 label.pack()  
  
 **if** flag == 1:  
 first\_step = xor(blocksBin[0], codingDES(vector\_init, 1, keys))  
 block64b = first\_step  
 block64Int = b''  
 **for** i **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **for** p **in** range(1, len(blocksBin)):  
 first\_step = xor(blocksBin[p], codingDES(first\_step, 1, keys))  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass** block64b = first\_step  
 **for** i **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **else**:  
 first\_step = xor(blocksBin[0], codingDES(vector\_init, 1, keys))  
 block64b = first\_step  
 block64Int = b''  
 **for** i **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 **for** p **in** range(1, len(blocksBin)):  
 first\_step = xor(blocksBin[p], codingDES(blocksBin[p-1], 1, keys))  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass** block64b = first\_step  
 **for** i **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
  
 blocksText = after\_coding(file, block64Int, flag)  
 **if** normal == 0:  
 root.destroy()  
 **return** blocksText  
  
**def** OFB(keys1, text, flag: int, vector\_init:str):  
 **if not** vector\_init **or** vector\_init == '':  
 vector\_init = '1110011110010101110010111001010010000111001010011100101001001011' #КАКОЙ БРАТЬ?  
 **else**:  
 **if** len(vector\_init) != 64:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** vector\_init:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в векторе инициализации!!")  
 msgBox.exec\_()  
 **return** ("")  
 **if not** keys1 **or** keys1 == '':  
 keys = genKeys('DESkey56')  
 **else**:  
 **if** len(keys1) != 16 \* 48:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 **for** i **in** keys1:  
 **if** (i != '1' **and** i != 1 **and** i != '0' **and** i != 0) == **True**:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Ошибка в ключе!")  
 msgBox.exec\_()  
 **return** ("")  
 keys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
 **for** i **in** range(16):  
 keys[i] = keys1[:48]  
 keys1 = keys1[48:]  
 **if not** text:  
 msgBox = QtWidgets.QMessageBox()  
 msgBox.setWindowTitle("Ошибка")  
 msgBox.setText("Введите текст!")  
 msgBox.exec\_()  
 **return** ("")  
  
 blocksBin, file = before\_coding(text, flag)  
  
 percent = 100 / len(blocksBin)  
 progress = 0  
 normal = 0  
 root = Tk()  
  
 label = Label(text='Progressbar')  
 x = (root.winfo\_screenwidth() - root.winfo\_reqwidth()) / 2  
 y = (root.winfo\_screenheight() - root.winfo\_reqheight()) / 2  
 root.wm\_geometry("+%d+%d" % (x, y))  
 label.pack()  
  
  
 block64Int = b''  
 first\_step = codingDES(vector\_init, 1, keys)  
 **for** p **in** range(len(blocksBin)):  
 block64b = xor(first\_step, blocksBin[p])  
 **try**:  
 progress += percent  
 label.config(text='**\r**Обработка завершена на %3d%%' % progress)  
 root.update()  
 # print('\rОбработка завершена на %3d%%' % progress, end = '', flush = True)  
 time.sleep(0.01)  
 **except** Exception:  
 normal = 1  
 **pass  
 for** i **in** range(8):  
 block64Int+=bytes([int(block64b[:8], 2)])  
 block64b = block64b[8:]  
 first\_step = codingDES(first\_step, 1, keys)  
  
 blocksText = after\_coding(file, block64Int, flag)  
 **if** normal == 0:  
 root.destroy()  
 **return** blocksText

1. **Примеры работы программы**

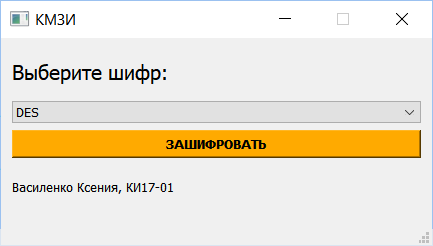


Рисунок 8 – Главное окно.

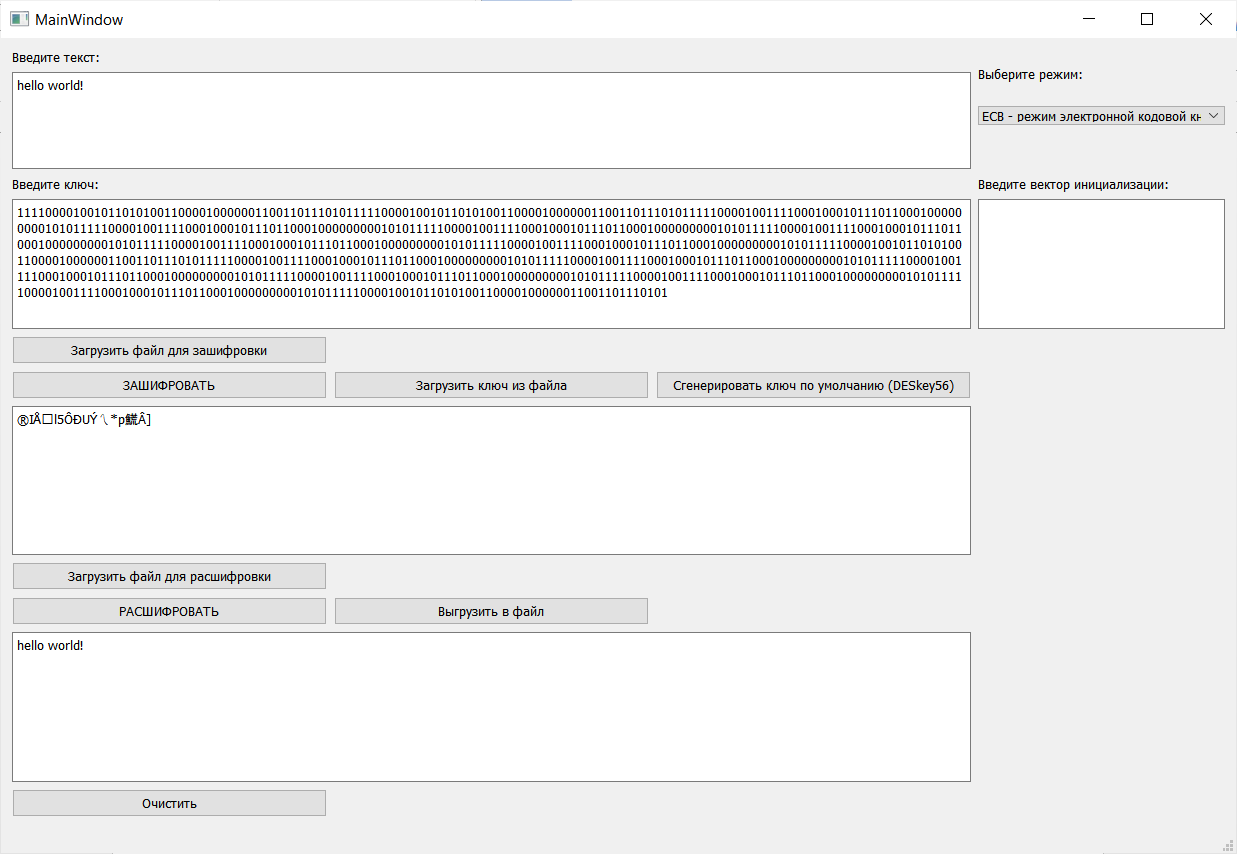


Рисунок 9 – Шифр DES - ECB.

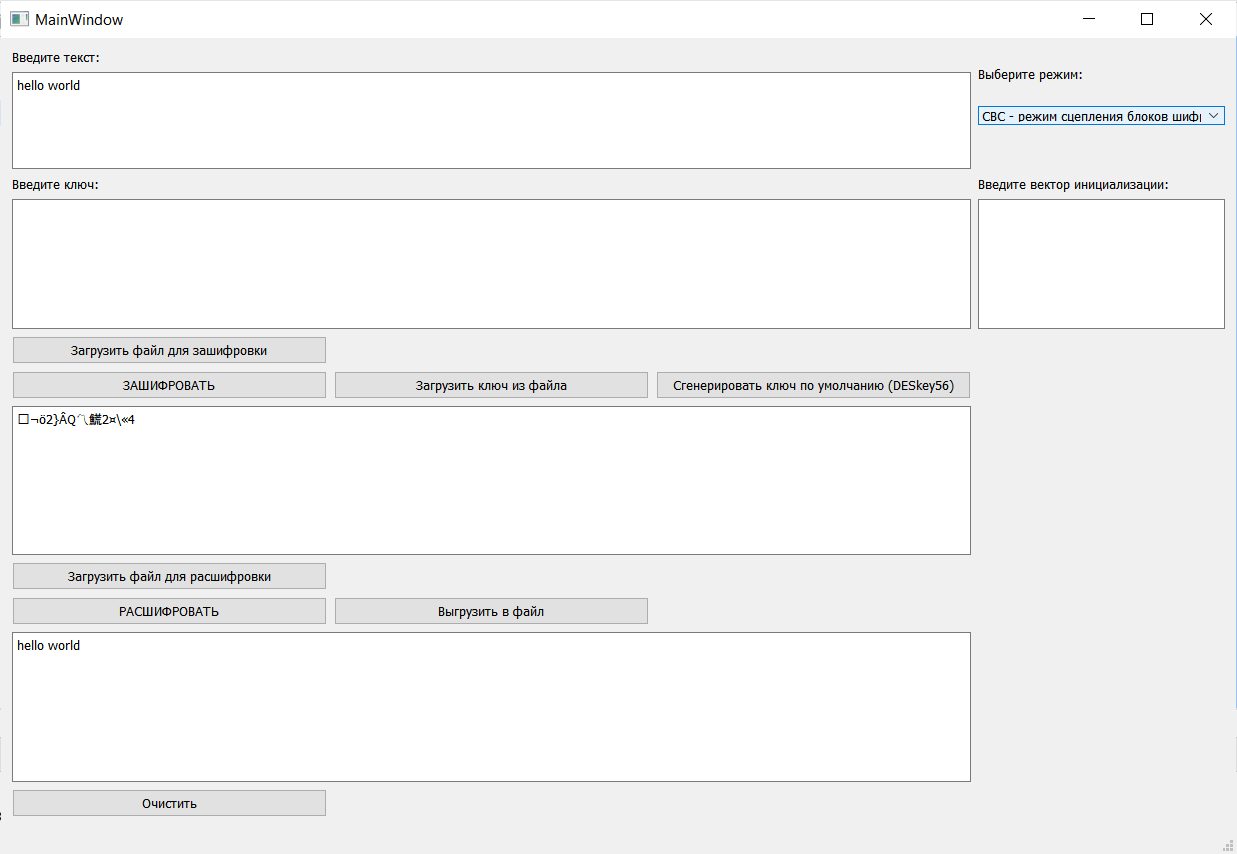


Рисунок 10 – Шифр DES – CBC.

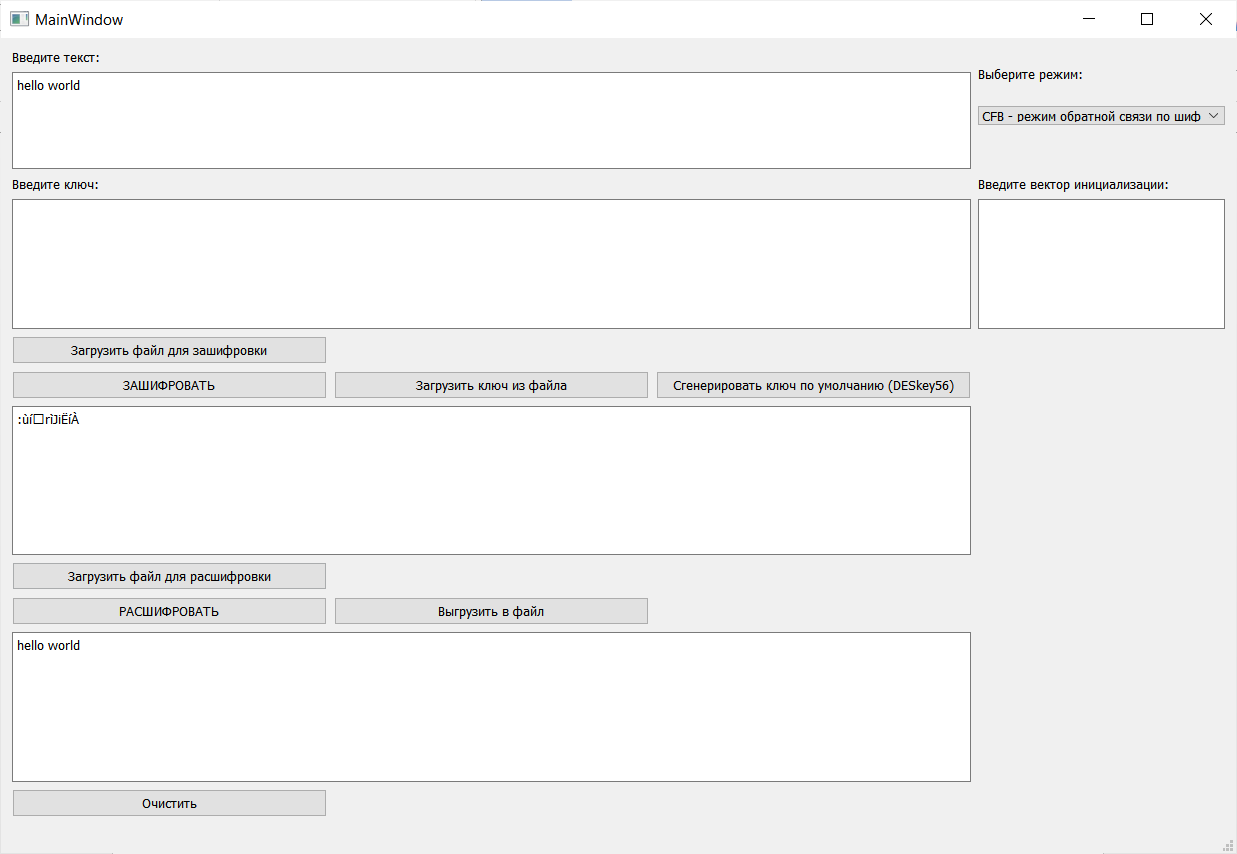


Рисунок 11 – Шифр DES – CFB.

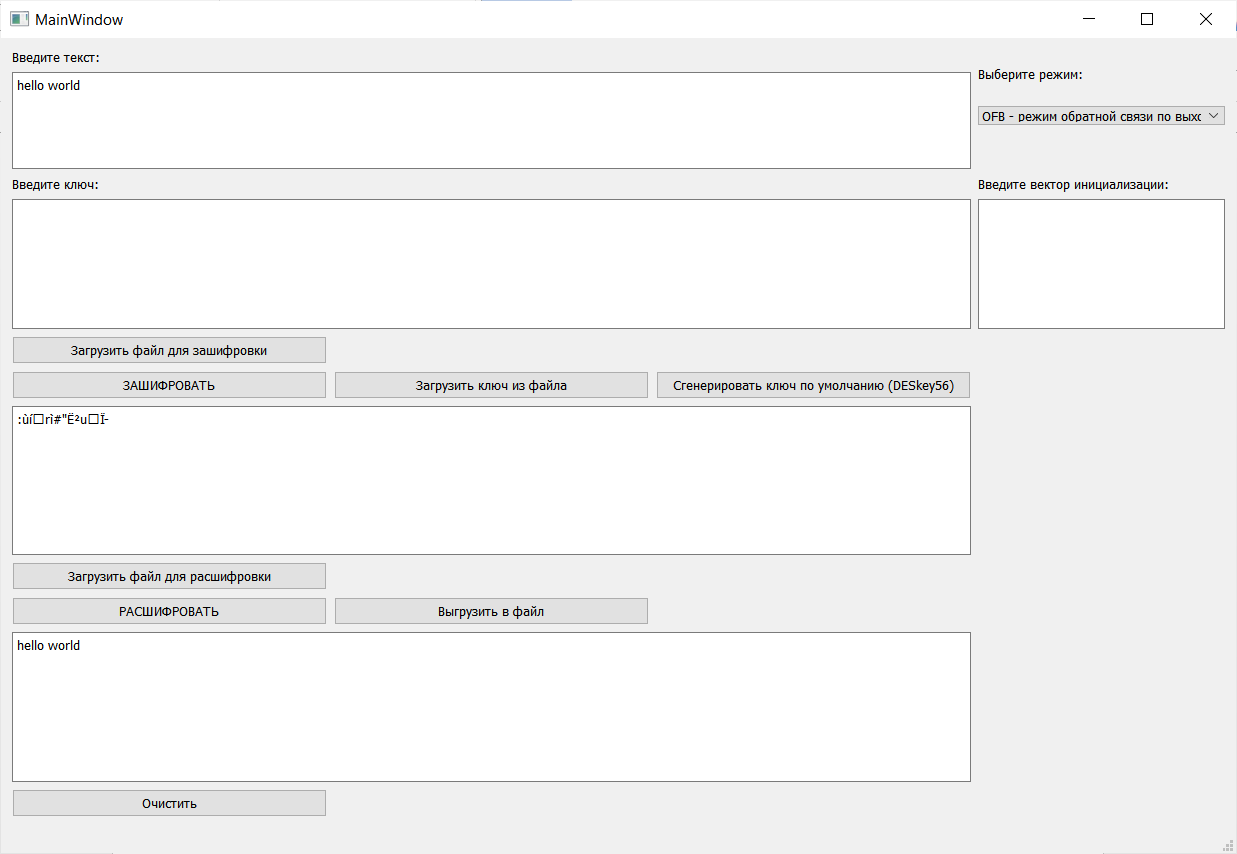
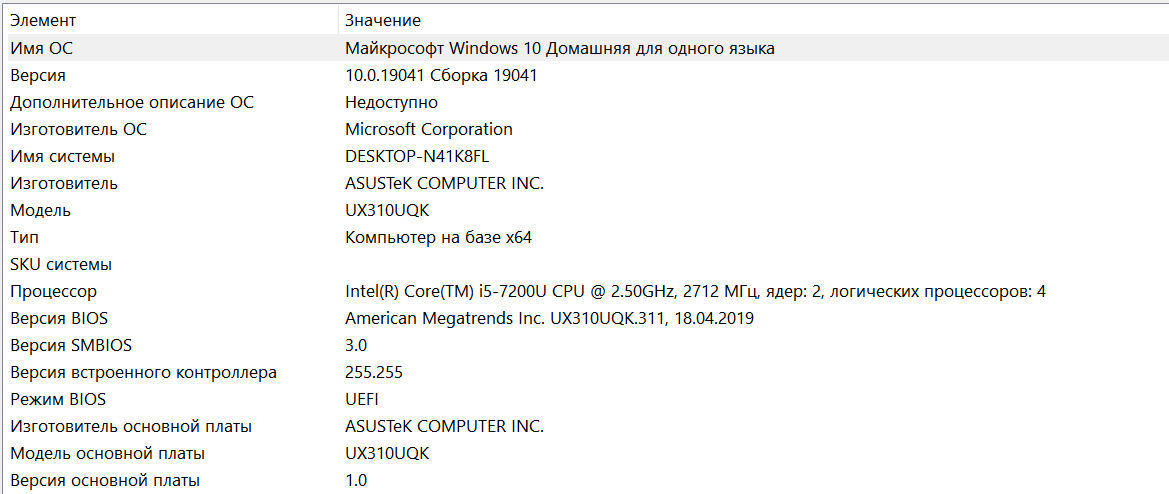


Рисунок 12 – Шифр DES – OFB.

1. **Тестирование**

Характеристики:



Файл 2 МБ обрабатывается (и зашифровка, и расшифровка) в режиме ECB примерно 2 часа.

# ЗАКЛЮЧЕНИЕ

Во время выполнения лабораторной работы №16 мною был изучен и реализован программно шифр DES, а также предусмотрен графический интерфейс с помощью PyQt5 и Qt designer.