Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №2

по курсу «Защита информации»

на тему: «Алгоритм шифрования DES»

Студент ИУ7-71Б _____ Постнов С. А.
(Группа) (Подпись, дата) (Фамилия И. О.)

Преподаватель _____ Чиж И. С.
(Подпись, дата) (Фамилия И. О.)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является реализация в виде программы шифровального алгоритма DES в режиме работы CBF.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать шифровальный алгоритм DES;

2) описать режим работы CBF;

3) определить средства программной реализации;

4) реализовать программу, реализующую алгоритм шифрования DES;

5) протестировать реализованную программу.

1 Аналитический раздел

1.1 Алгоритм DES

Шифровальная алгоритм DES — симметричный шифровальный алгоритм, разработанный в 1977 году компанией IBM. Он использует блочное шифрование, длина блока фиксирована и равна 64 битам. Однако каждые 8 бит в ключе игнорируются, что приводит к правильной длине ключа 56 бит в DES. Однако в любом случае один блок на 64 бита является вечной организацией DES. Он состоит из 3 следующих шагов:

- начальная перестановка, во время которой биты переставляются в порядке, определённом в специальной таблице;
- 16 раундов шифрования;
- завершающей перестановки, соовершающей преобразования, обратные сделанным на первом шаге.

Раунд шифрования состоит из 5 следующих этапов:

1) расширение;
2) получение ключа раунда;
3) скремблирование;
4) перестановка;
5) смешивание ключа.

Расширение, во время которого каждая из половин блока шифрования по 32 бит дополняется путём перестановки и дублировоания бит до длины в 48 бит.

Получение ключа раунда необходимо для применения в раунде шифрования 48-битного ключа раунда, полученного из основного ключа DES. Основной ключ имеет длину 64 бита, однако значащих бит из 64 всего 56, остальные добавлены для избыточности и контроля передачи ключа. Из этих 56 бит получают 48 путём разбиения на равные части и применению битовой

операции циклического сдвига и нахождению нового значения посредством специальной таблицы.

Скремблирование предназначено для получения из 48-битного потока 32-битного путём разбиения на 6 частей по 8 бит и обработки каждой части в S-блоках, которые заменяют блоки с длиной 6 бит на блоки 4 бит посредством использования специальной таблицы.

Перестановка представляет из себя перемешивания полученной последовательности из 32 бит при помощи таблицы перемешивания.

Смешивание ключа представляет из себя операцию XOR полученного 32-битного значения с ключом раунда.

1.2 Режимы работы алгоритма DES

Режим шифрования — метод применения блочного шифра, позволяющий преобразовать последовательность блоков открытых данных в последовательность блоков зашифрованных данных.

Наиболее широкое распространение получили режимы [1]:

1) электронный шифроблокнот (Electronic Codebook) - ECB;
2) цепочка цифровых блоков (Cipher Block Chaining) - CBC;
3) цифровая обратная связь (Cipher Feedback) - CFB;
4) внешняя обратная связь (Output Feedback) - OFB.

1.3 CBF

В этом режиме размер блока может отличаться от 64. Исходный файл M считывается последовательными t-битовыми блоками (t <= 64): M = M(1)M(2)...M(n) (остаток дописывается нулями или пробелами). 64-битовый сдвиговый регистр (входной блок) вначале содержит вектор инициализации IV, выравненный по правому краю. Для каждого сеанса шифрования используется новый IV. Для всех i = 1...n блок шифртекста C(i) определяется следующим образом:

C(i) = M(i) xor P(i-1) ,

где P(i-1) - старшие t битов операции DES(C(i-1)), причем C(0)=IV. Обновление сдвигового регистра осуществляется путем удаления его старших t битов и дописывания справа C(i). Восстановление зашифрованных данных также не представляет труда: P(i-1) и C(i) вычисляются аналогичным образом и M(i) = C(i) xor P(i-1) [1].

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунке 2.1 представлена обобщенная схема шифрования в алгоритме DES.

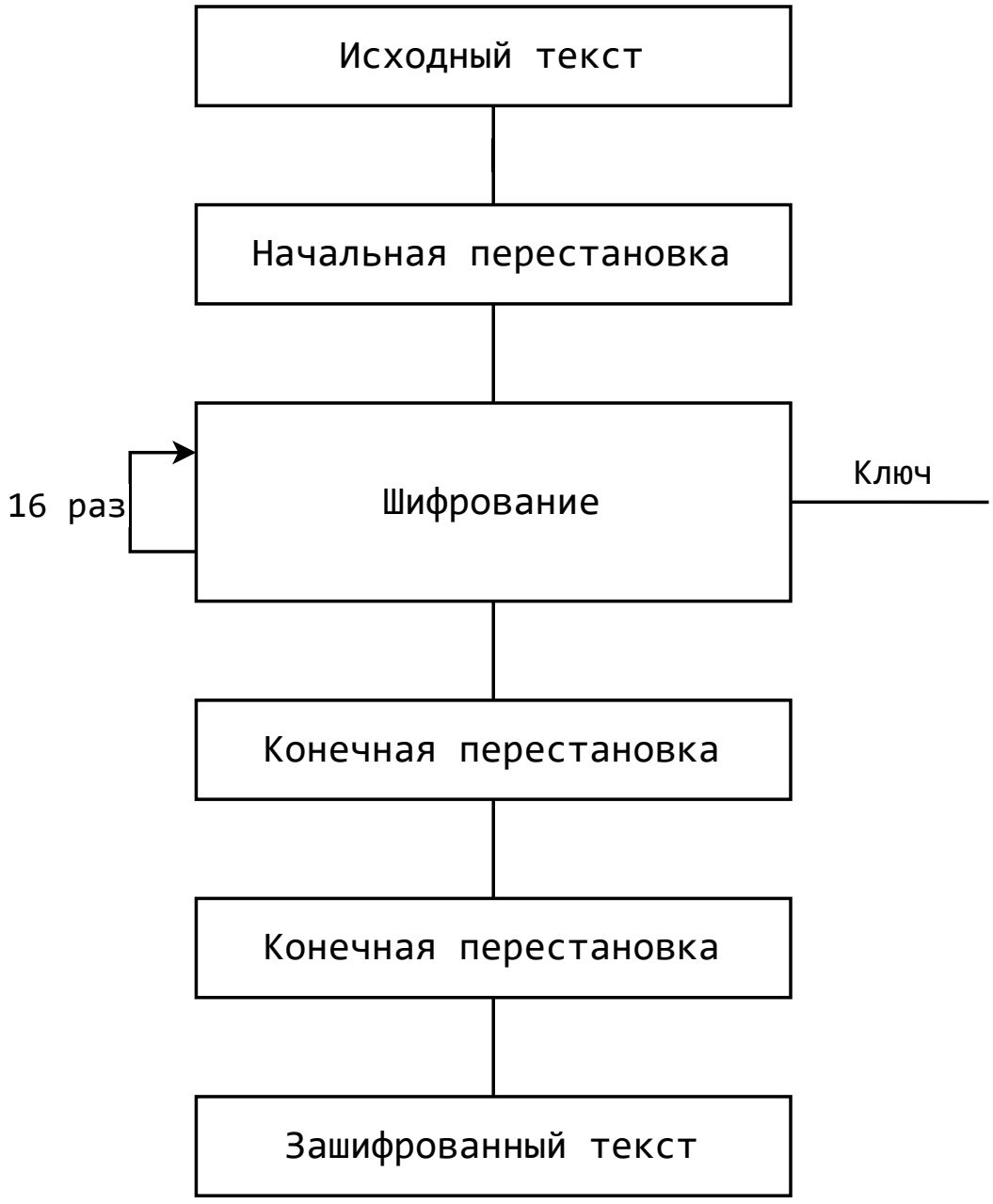

Рисунок 2.1 – Обобщенная схема шифрования в алгоритме DES

На рисунке 2.2 представлен алгоритм всех раундов.

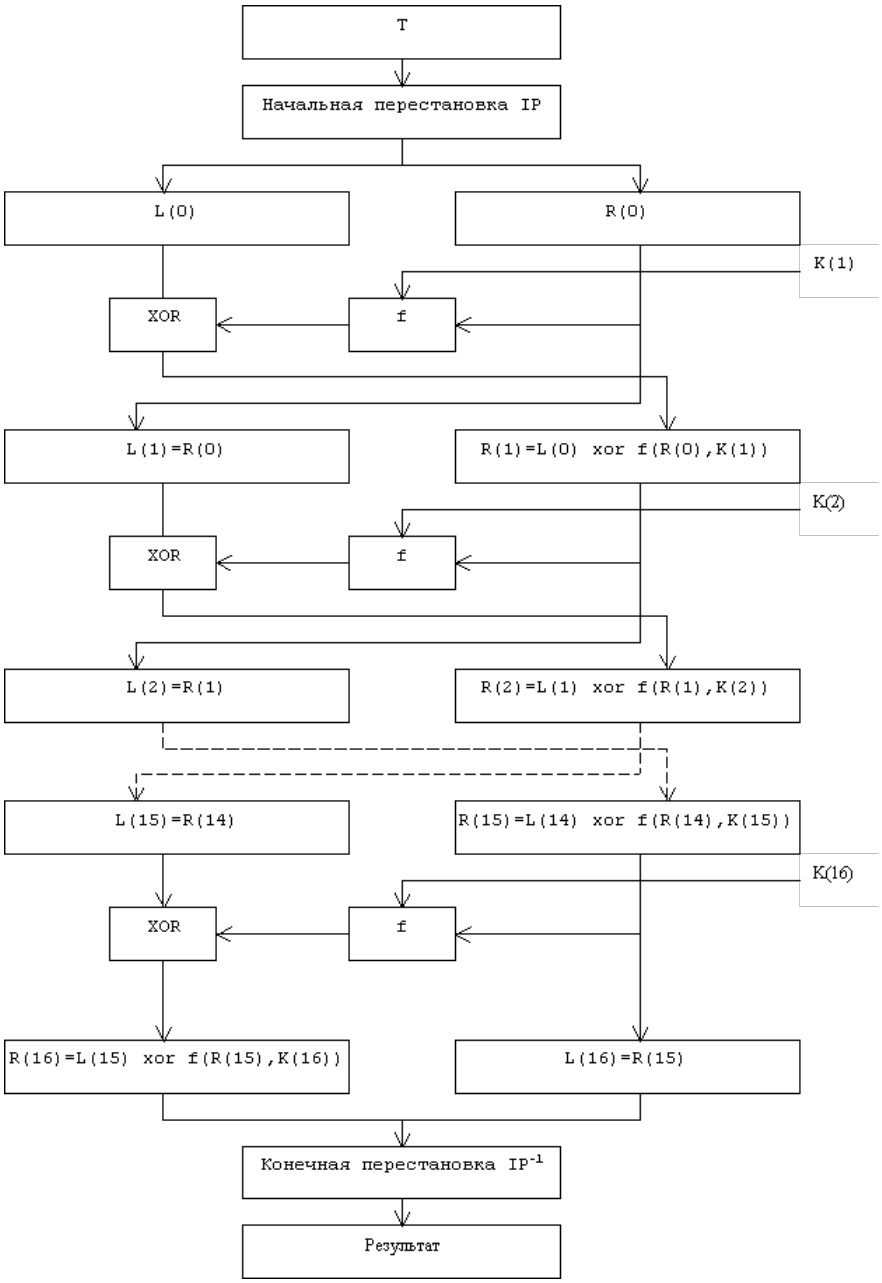

Рисунок 2.2 – Схема алгоритма всех раундов

На рисунке 2.3 представлен алгоритм функции Фейстеля.

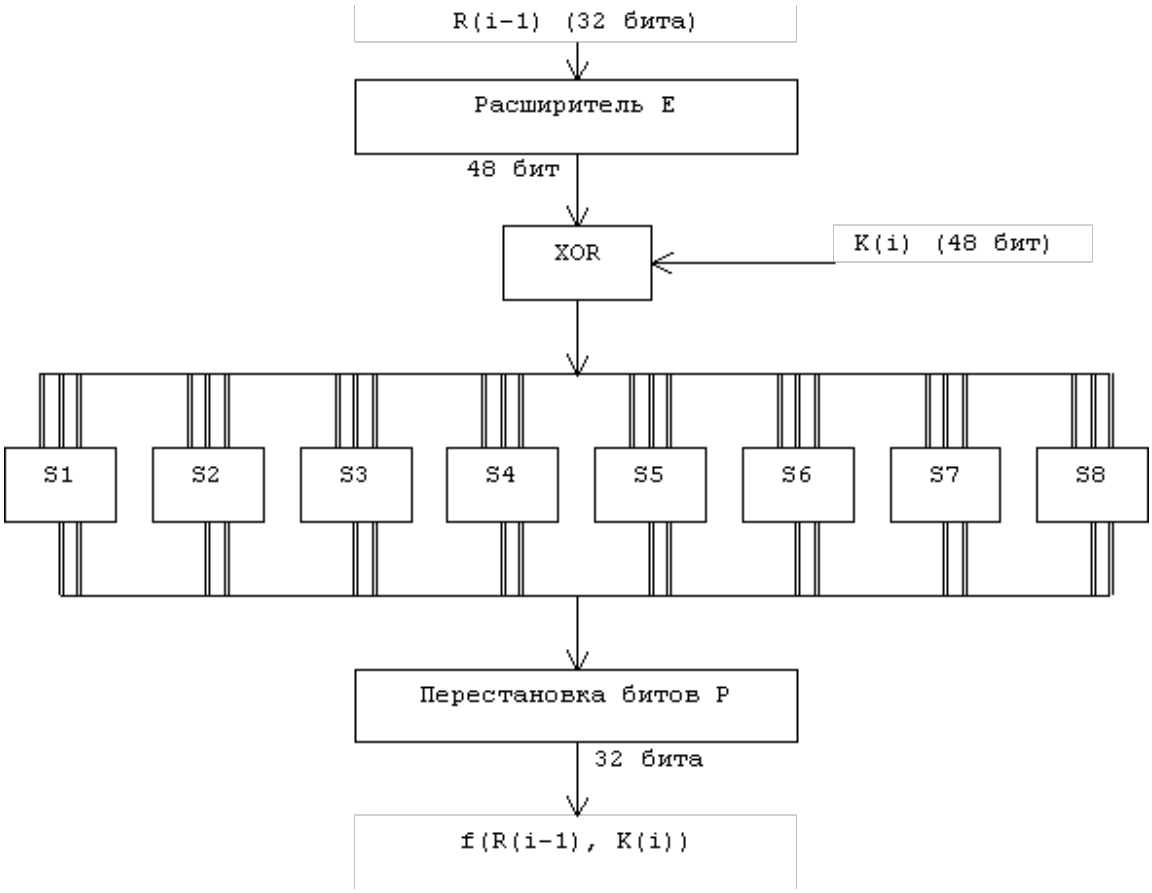

Рисунок 2.3 – Схема алгоритма функции Фейстеля

На рисунке 2.4 представлен алгоритм режима работы CFB

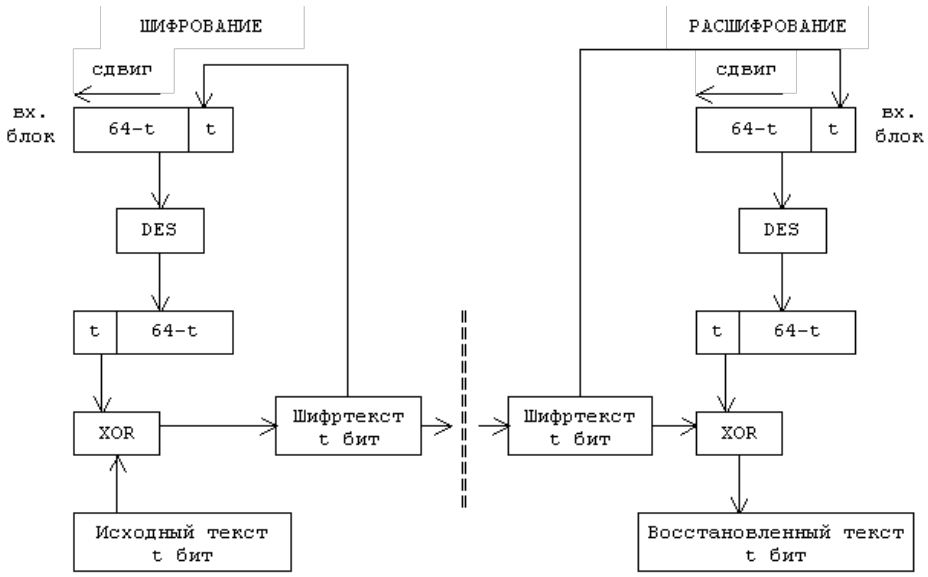

Рисунок 2.4 – Работа алгоритма в режиме CFB

3 Технологический раздел

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [1].

3.2 Реализация алгоритмов

На листингах 3.1 – 3.2 представлены реализации разрабатываемых модулей.

Листинг 3.1 – Реализация класса алгоритма DES

```
#pragma once

#include <fstream>
#include <vector>

using namespace std;

typedef bitset<64> bitset64;
typedef bitset<56> bitset56;
typedef bitset<48> bitset48;
typedef bitset<32> bitset32;
typedef bitset<28> bitset28;

class DES {
public:
    bitset64 encrypt(bitset64 plaintext, bitset64 key) {
        vector<bitset48> keys = generate_keys(key);

        bitset64 permuted_text = permute<64, 64>(plaintext,
           initial_permutation, 64);

        bitset32 L = (permuted_text >> 32).to_ulong();
        bitset32 R = permuted_text.to_ulong();

        for (int i = 0; i < 16; ++i) {
            bitset32 temp = R;
            R = L ^ function_F(R, keys[i]);
            L = temp;
        }
```

```
29
30          bitset64 preoutput = (bitset64(R.to_ulong()) << 32) |
                bitset64(L.to_ulong());
31
32          return permute<64, 64>(preoutput, final_permutation, 64);
33      }
34  private:
35      int initial_permutation[64] = {
36          58, 50, 42, 34, 26, 18, 10,  2,
37          60, 52, 44, 36, 28, 20, 12,  4,
38          62, 54, 46, 38, 30, 22, 14,  6,
39          64, 56, 48, 40, 32, 24, 16,  8,
40          57, 49, 41, 33, 25, 17,  9,  1,
41          59, 51, 43, 35, 27, 19, 11,  3,
42          61, 53, 45, 37, 29, 21, 13,  5,
43          63, 55, 47, 39, 31, 23, 15,  7
44      };
45
46      int final_permutation[64] = {
47          40,  8, 48, 16, 56, 24, 64, 32,
48          39,  7, 47, 15, 55, 23, 63, 31,
49          38,  6, 46, 14, 54, 22, 62, 30,
50          37,  5, 45, 13, 53, 21, 61, 29,
51          36,  4, 44, 12, 52, 20, 60, 28,
52          35,  3, 43, 11, 51, 19, 59, 27,
53          34,  2, 42, 10, 50, 18, 58, 26,
54          33,  1, 41,  9, 49, 17, 57, 25
55      };
56
57      int gen_keys_g[64] = {
58          57, 49, 41, 33, 25, 17, 9 ,
59          1 , 58, 50, 42, 34, 26, 18,
60          10, 2 , 59, 51, 43, 35, 27,
61          19, 11, 3 , 60, 52, 44, 36,
62          63, 55, 47, 39, 31, 23, 15,
63          7 , 62, 54, 46, 38, 30, 22,
64          14, 6 , 61, 53, 45, 37, 29,
65          21, 13, 5 , 28, 20, 12, 4
66      };
67
68      int gen_keys_h[64] = {
```

```
        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
    };

    int expansion_table[48] = {
        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
    };

    int S[8][4][16] = {
        {
            {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}
        },
        {
            {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}
```

```
        },
        {
            {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}
        },
        {
            {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}
        },
        {
            {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}
        },
        {
            {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}
        },
```

```
127         {
128             {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6,
                    1},
129             {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8,
                    6},
130             {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9,
                    2},
131             {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3,
                    12}
132         },
133         {
134             {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12,
                    7},
135             {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9,
                    2},
136             {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5,
                    8},
137             {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6,
                    11}
138         }
139     };
140
141     int permutation_P[32] = {
142         16, 7, 20, 21,
143         29, 12, 28, 17,
144         1, 15, 23, 26,
145         5, 18, 31, 10,
146         2, 8, 24, 14,
147         32, 27, 3, 9,
148         19, 13, 30, 6,
149         22, 11, 4, 25
150     };
151
152     int shift_schedule[16] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,
            2, 2, 2, 2, 1 };
153     std::vector<char> key;
154
155     template <size_t InputSize, size_t OutputSize>
156     bitset<OutputSize> permute(bitset<InputSize> input, const
            int* table, int n) {
157         bitset<OutputSize> output;
```

```
158         for (int i = 0; i < n; ++i) {
159             output[n - 1 - i] = input[InputSize - table[i]];
160         }
161         return output;
162     }
163 
164     bitset48 expansion(bitset32 input) {
165         bitset48 output;
166 
167         for (int i = 0; i < 48; ++i)
168             output[47 - i] = input[32 - expansion_table[i]];
169 
170         return output;
171     }
172 
173     bitset32 function_F(bitset32 R, bitset48 K) {
174         bitset48 ER = expansion(R);
175         bitset48 B = ER ^ K;
176         bitset32 result;
177 
178         for (int i = 0; i < 8; ++i) {
179             int row = (B[47 - 6 * i] << 1) | B[47 - 6 * i - 5];
180             int col = (B[47 - 6 * i - 1] << 3) | (B[47 - 6 * i -
                2] << 2) | (B[47 - 6 * i - 3] << 1) | B[47 - 6 *
                i - 4];
181             int sbox_value = S[i][row][col];
182 
183             for (int j = 0; j < 4; ++j)
184                 result[31 - 4 * i - j] = (sbox_value >> j) & 1;
185         }
186 
187         return permute<32, 32>(result, permutation_P, 32);
188     }
189 
190     vector<bitset48> generate_keys(bitset64 key) {
191         vector<bitset48> keys;
192         bitset56 permuted_key = permute<64, 56>(key, gen_keys_g,
            56);
193 
194         bitset28 C = (permuted_key >> 28).to_ulong();
195         bitset28 D = permuted_key.to_ulong();
```

```
196
197         for (int i : shift_schedule) {
198             C = (C << i) | (C >> (28 - i));
199             D = (D << i) | (D >> (28 - i));
200
201             bitset56 combined_key = (bitset56(C.to_ulong()) <<
                    28) | bitset56(D.to_ulong());
202
203             bitset48 round_key = permute<56, 48>(combined_key,
                    gen_keys_h, 48);
204             keys.push_back(round_key);
205         }
206
207         return keys;
208     }
209 };
```

Листинг 3.2 – Реализация класса шифровального алгоритма DES в режиме работы CFB

```
1  #pragma once
2
3  #include <iostream>
4
5  #include "des.hpp"
6
7  using namespace std;
8
9  typedef bitset<64> bitset64;
10 typedef bitset<56> bitset56;
11 typedef bitset<48> bitset48;
12 typedef bitset<32> bitset32;
13 typedef bitset<28> bitset28;
14
15 class CBF {
16 public:
17     template<size_t BlockSize>
18     vector<bitset<BlockSize>> encrypt(const
          vector<bitset<BlockSize>>& plaintext_blocks, const
          bitset64 key, const bitset64 IV) {
19         vector<bitset<BlockSize>> ciphertext_blocks;
20         bitset64 feedback = IV;
21
```

```
22          for (auto &block : plaintext_blocks) {
23              bitset64 encrypted_feedback =
                    DES().encrypt(feedback, key);
24              bitset<BlockSize> extended_encrypted_feedback =
                    bitset<BlockSize>(encrypted_feedback.to_ullong());
25
26              bitset<BlockSize> ciphertext_block = XOR(block,
                    extended_encrypted_feedback);
27              ciphertext_blocks.push_back(ciphertext_block);
28
29              feedback = (feedback << BlockSize) |
                    bitset64(ciphertext_block.to_ullong());
30          }
31
32          return ciphertext_blocks;
33      }
34
35      template<size_t BlockSize>
36      vector<bitset<BlockSize>> decrypt(const
            vector<bitset<BlockSize>>& ciphertext_blocks, const
            bitset64 key, const bitset64 IV) {
37          vector<bitset<BlockSize>> decrypted_blocks;
38          bitset64 feedback = IV;
39
40          for (auto &block : ciphertext_blocks) {
41              bitset64 encrypted_feedback =
                    DES().encrypt(feedback, key);
42              bitset<BlockSize> extended_encrypted_feedback =
                    bitset<BlockSize>(encrypted_feedback.to_ullong());
43
44              bitset<BlockSize> plaintext_block = XOR(block,
                    extended_encrypted_feedback);
45              decrypted_blocks.push_back(plaintext_block);
46
47              feedback = (feedback << BlockSize) |
                    bitset64(block.to_ullong());
48          }
49
50          return decrypted_blocks;
51      }
52
```

```
53      template<size_t BlockSize>
54      void encryptFile(const std::string& inputFile, const
           std::string& outputFile, bitset64 IV, bitset64 key, char
           t) {
55          ifstream in;
56          ofstream out(outputFile);;
57
58          if (t == 't')
59              in = ifstream(inputFile);
60          else
61              in = ifstream(inputFile, ios::binary);
62
63          if (!in.is_open() || !out.is_open()) {
64              throw runtime_error("Не удалось открыть файл.");
65          }
66
67          vector<bitset<BlockSize>> blocks;
68          long blockSizeBytes = BlockSize / 8;
69          vector<char> buffer(blockSizeBytes);
70
71          while (in.read(buffer.data(), blockSizeBytes))
72              blocks.push_back(charArrayToBitset<BlockSize>(buffer,
                   blockSizeBytes));
73
74          if (in.gcount() > 0) {
75              memset(buffer.data() + in.gcount(), 0,
                   blockSizeBytes - in.gcount());
76              blocks.push_back(charArrayToBitset<BlockSize>(buffer,
                   blockSizeBytes));
77          }
78
79          auto encrypted_blocks = encrypt(blocks, key, IV);
80
81          for (const auto& block : encrypted_blocks)
82              if (t == 't')
83                  out << bitsetToASCII(block);
84              else
85                  out.write(bitsetToBytes(block).data(),
                       bitsetToBytes(block).size());
86
87          in.close();
```

```
88          out.close();
89      }
90
91      template<size_t BlockSize>
92      void decryptFile(const std::string& inputFile, const
           std::string& outputFile,  bitset64 IV, bitset64 key, char
           t) {
93          ifstream in;
94          ofstream out(outputFile);;
95
96          if (t == 't')
97              in = ifstream(inputFile);
98          else
99              in = ifstream(inputFile, ios::binary);
100
101         if (!in.is_open() || !out.is_open()) {
102             throw runtime_error("Не удалось открыть файл.");
103         }
104
105         vector<bitset<BlockSize>> blocks;
106         long blockSizeBytes = BlockSize / 8;
107         vector<char> buffer(blockSizeBytes);
108
109         while (in.read(buffer.data(), blockSizeBytes))
110             blocks.push_back(charArrayToBitset<BlockSize>(buffer,
                   blockSizeBytes));
111
112         auto decrypted_blocks = decrypt(blocks, key, IV);
113
114         for (const auto& block : decrypted_blocks)
115             if (t == 't')
116                 out << bitsetToASCII(block);
117             else
118                 out.write(bitsetToBytes(block).data(),
                       bitsetToBytes(block).size());
119
120         in.close();
121         out.close();
122     }
123 private:
124     template<size_t BlockSize>
```

```
125     static bitset<BlockSize> XOR(bitset<BlockSize> a,
            bitset<BlockSize> b) {
126         return a ^ b;
127     }
128
129     template<size_t BlockSize>
130     bitset<BlockSize> charArrayToBitset(vector<char> array,
            const size_t size) {
131         uint64_t value = 0;
132
133         for (size_t i = 0; i < size; ++i)
134             value = (value << 8) | static_cast<unsigned
                    char>(array[i]);
135
136         return bitset<BlockSize>(value);
137     }
138
139     template<size_t N>
140     std::string bitsetToASCII(const std::bitset<N>& b) {
141         std::string asciiString;
142
143         for (int i = N - 1; i >= 0; i -= 8) {
144             char c = 0;
145
146             for (int j = 0; j < 8; ++j)
147                 c = (c << 1) | b[i - j];
148
149             asciiString += c;
150         }
151
152         return asciiString;
153     }
154
155     template<size_t N>
156     std::vector<char> bitsetToBytes(const std::bitset<N>& b) {
157         std::vector<char> bytes(N / 8);
158
159         for (int i = 0; i < N; ++i)
160             for (int j = 0; j < 8; ++j)
161                 bytes[i] = (bytes[i] << 1) | b[N - 1 - i * 8 -
                        j];
```

```
162
163             return bytes;
164         }
165 };
```

3.3 Тестирование

В таблице 3.1 представлены функциональные тесты.

Таблица 3.1 – Функциональные тесты

№	Входные данные	Выходные данные
1	пустая строка	пустая строка
2	пустой файл	пустой файл
3	aaaaa	зашифрованный "aaaaa"
4	зашифрованный "aaaaa"	aaaaa
5	abcde	зашифрованный "abcde"
6	зашифрованный "abcde"	abcde
7	файл input.txt	зашифрованный файл input.txt
8	зашифрованный файл input.txt	файл input.txt
9	файл input.jpg	зашифрованный файл input.jpg
10	зашифрованный файл input.jpg	файл input.jpg
11	файл input.zip	зашифрованный файл input.zip
12	зашифрованный файл input.zip	файл input.zip

ЗАКЛЮЧЕНИЕ

Цель, поставленная в начале работы, была достигнута. Кроме того были достигнуты все поставленные задачи:

1) описан шифровальный алгоритм DES;

2) описан режим работы CFB;

3) определены средства программной реализации;

4) реализована программа, реализующая алгоритм шифрования DES;

5) протестирована реализованная программу.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. C++ language. — [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/language` (дата обращения: 16.10.2024).