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: «DES»

Студент ИУ7-73Б _____ Лагутин Д. В.

(Группа) (Подпись, дата) (Фамилия И. О.)

Преподаватель _____ Чиж И. С.

(Подпись, дата) (Фамилия И. О.)

Москва, 2023 г.

Содержание

Введение

Целью работы является реализация программы шифрования симметричным алгоритмом DES с применением режима шифрования PCBC (вариант 3). Необходимо обеспечить шифрование и расшифровку произвольного файла с использованием разработанной программы. Предусмотреть работу программы с пустым, однобайтовым файлом. Программа также должна уметь обрабатывать файл архива (rar, zip или др.).

Задачи:

1) изучить алгоритм шифрования и применяемый режим;
2) разработать алгоритм работы программы;
3) реализовать и протестировать разработанное программное обеспечение.

1 Аналитическая часть

1.1 Описание алгоритма

DES (Data Encryption Standard) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт.

Алгоритм является блочным и основывается на применении сетей Фейстеля. Открытый текст разбивается на блоки размером 64 бита, в случае необходимости последний блок дополняется нулями. Применяемый ключ также имеет размер 64 бита, однако каждый 8 бит является битом четности и не участвует в шифровании (фактический размер 56 бит).

Шифрование каждого блока состоит из следующих шагов:

— производится первоначальная перестановка бит сообщения (блок IP);

— полученный результат разбивается на 2 блока по 32 бита (L_0 и R_0 соответственно;

— выполняется побитовый xor левой части с результатом работы функции Фейстеля для ключа и правой части;

— результат предыдущего шага записывается в правую ячейку, правая же ячейка попадает в левую без изменений;

— 2 предыдущих шага (раунд) выполняются 16 раз

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus \mathrm{f}(R_i, k_i);$$

— применяется перестановка, обратная первоначальной ($IP^{-}1$).

Расшифровка происходит в обратном порядке.

Функция Фейстеля состоит из следующих этапов:

— часть сообщения расширяется до 48 бит, за счет дополнения частей по 4 бита примыкающими битами;

— выполняется побитовый xor расширенного сообщения с i-ым ключом;

— полученный результат разделяется на 8 частей и поступает на вход соответствующим S-блокам, возвращающим 4 бита;

— фрагменты, полученный на предыдущем этапе, последовательно соединяются в итоговый результат.

S-блок является таблицей, состоящей из 4 строк и 16 столбцов, содержащей уникальные 4 битовые значения в пределах одной строки. Навигация по таблице происходит по следующим правилам:

— первый и последний биты определяют номер строки;

— средние — номер столбца.

Ключ для каждого раунда вырабатывается следующим образом:

— ключ разбивается на 2 половины согласно таблице;

— каждая половина циклически смещается влево, согласно значению таблицы на i-ом шаге;

— используя таблицу H, из половин составляется ключ для i-ого раунда шифрования.

1.2 Режим шифрования PCBC

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

Идея режима PCBC заключается в последовательном применении алгоритма DES с различными ключами к соответствующим блокам открытого текста, при этом перед шифровкой производится побитовый xor трех блоков: предыдущего блока открытого текста, соответствующего ему блока шифр текста и текущего блока открытого текста. Для первого блока используется значение вектора инициализации.

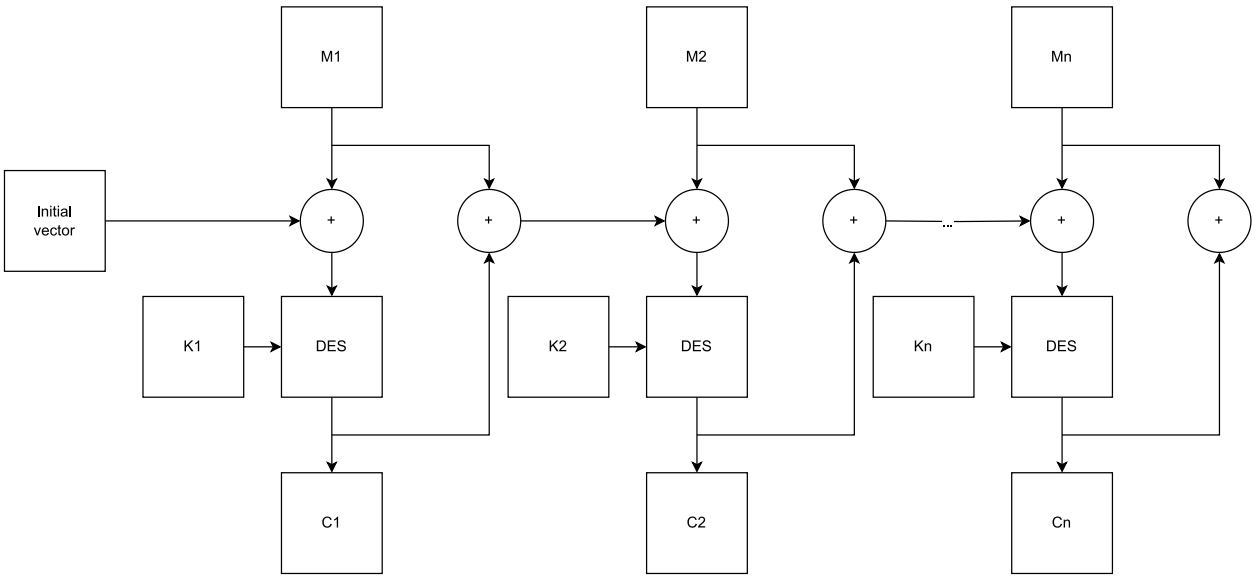

Рисунок 1.1 – Схема режима шифрования PCBC

Применение данного режима позволяет избежать статистических зависимостей при применении алгоритма DES. Отличительной особенностью данного метода является следующее свойство: если в каком-либо блоке шифр текста присутствует ошибка, то все последующие блоки также будут расшифрованы с ошибкой.

2 Конструкторская часть

2.1 Схема алгоритма работы

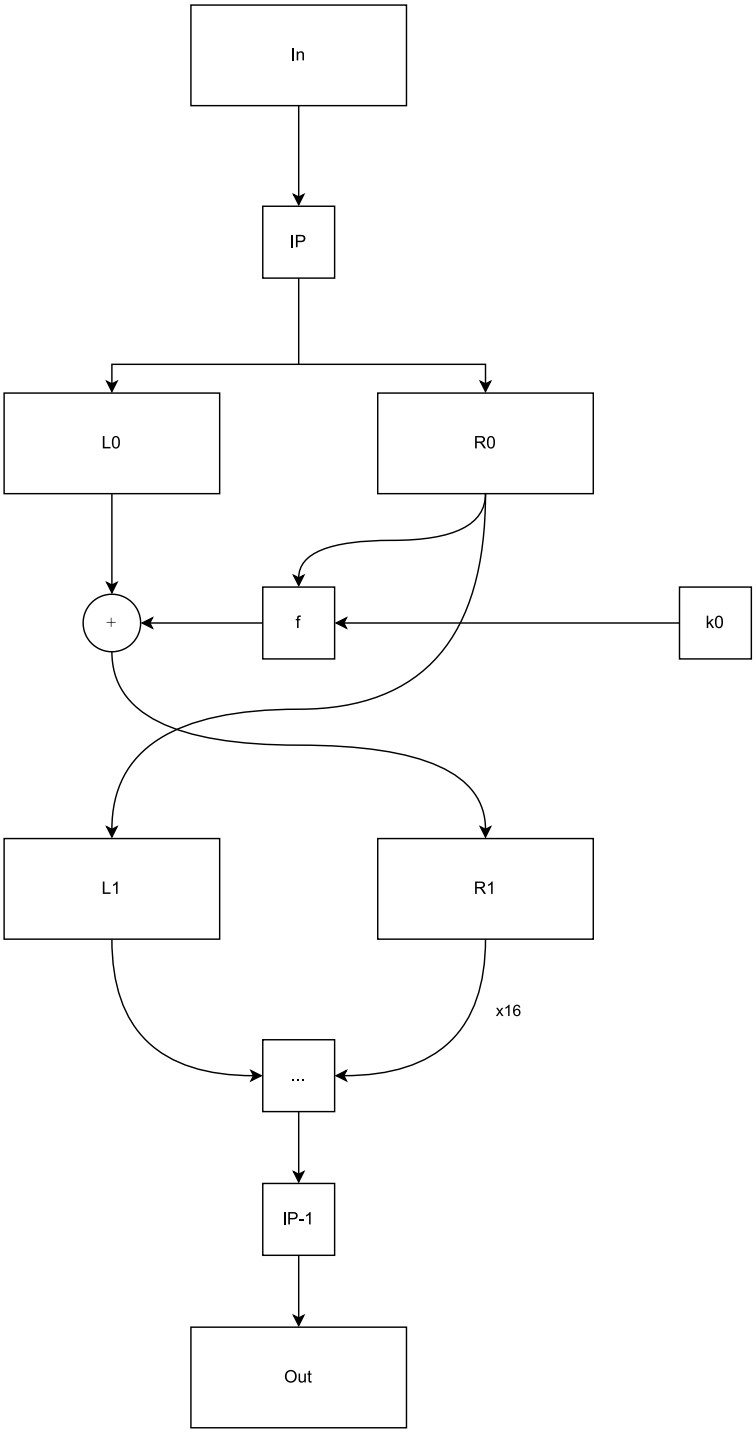

Рисунок 2.1 – Алгоритм работы DES

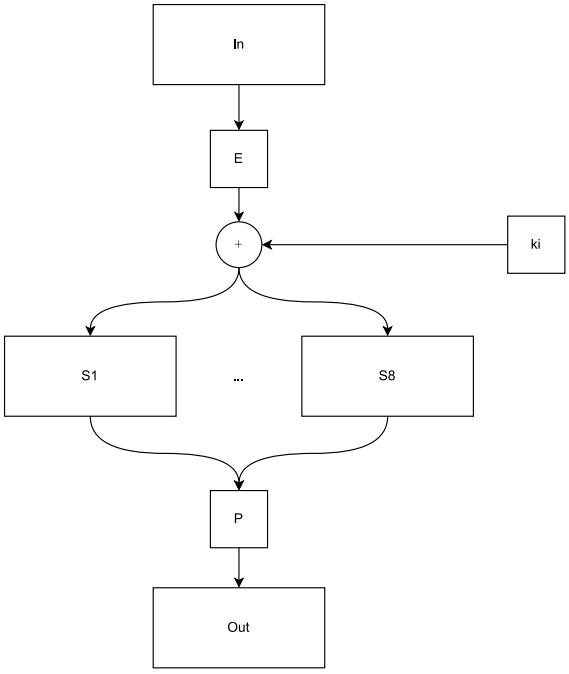

Рисунок 2.2 – Алгоритм ячейки Фейстеля

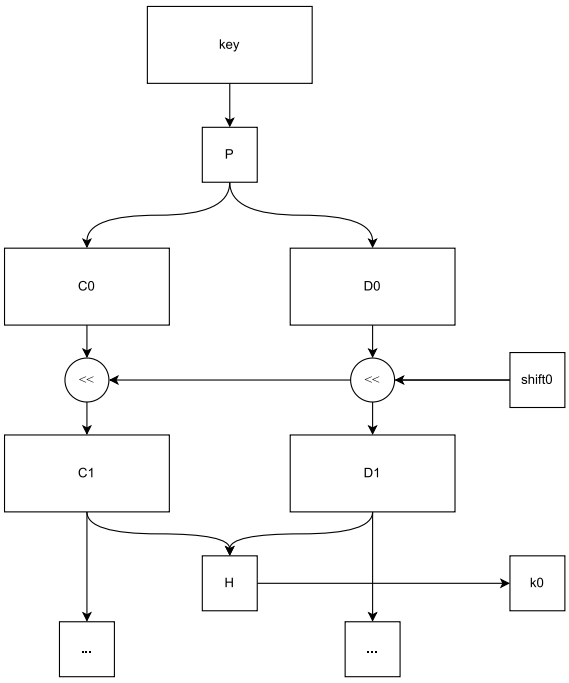

Рисунок 2.3 – Алгоритм получения ключа

3 Технологическая часть

3.1 Описание программного обеспечения

Для реализации машины использовался язык C++. Конфигурация осуществляется при помощи файла config.json, расположенного в корневом каталоге.

```
./config.json
{
    "initial_value": "./config/initial_value",
    "des_blocks": [
        "./config/des1/config.json",
        "./config/des2/config.json",
        "./config/des3/config.json"
    ]
}
```

```
./config/des1/config.json
{
    "f_block": {
        "path": "./f_block",
        "s_blocks": [
            "./s1", "./s2", "./s3", "./s4",
            "./s5", "./s6", "./s7", "./s8"
        ],
        "p_block": "./p_block"
    },

    "key_block": {
        "path": "./key_block",
        "key": "./key",
        "p_block": {
            "c": "./c_block",
            "d": "./d_block"
        },
        "shift_blocks": "./shift_blocks",
        "h_block": "./h_block"
    },

    "ip_block": "./ip_block"
}
```

Рисунок 3.1 – Пример конфигурации системы

Соответсвтующие концигурационный файлы состоят из массива типа `size_t`, за исключением начального вектора и ключа каждого отдельного блока DES.

Листинг 3.1 – Класс, реализующий алгоритм DES

```
1  DES::DES(std::shared_ptr<IPBlock> ip_block,
2           std::shared_ptr<FBlock> f_block,
3           std::shared_ptr<KeyBlock> key_block)
4      : ip_block(ip_block), f_block(f_block),
5        key_block(key_block)
6  {
7      if (nullptr == ip_block || nullptr == f_block
8          || nullptr == key_block)
9          throw;
10 }
11
12 std::string DES::encode(const std::string &origin)
13 {
14     size_t length = origin.length();
15
16     if (0 == length)
17         return origin;
18
19     size_t rest = length % 8;
20
21     if (0 != rest)
22         length += 8 - rest;
23
24     std::string out (length, 0);
25     char *buffer = out.data();
26
27     memmove(buffer, origin.c_str(), origin.length());
28
29     BitRange main_rng (buffer, 0, length * 8);
30
31     auto msgs = main_rng.splitSize(64);
32
33     for (size_t i = 0; msgs.size() > i; i++)
34         this->encodeBlock(msgs[i]);
35
36     return out;
37 }
38
39
40
```

```
41 std::string DES::decode(const std::string &origin)
42 {
43     size_t length = origin.length();
44
45     if (0 == length)
46         return origin;
47
48     if (0 != length % 8)
49         throw;
50
51     std::string out (length, 0);
52     char *buffer = out.data();
53
54     memmove(buffer, origin.c_str(), length);
55
56     BitRange main_rng (buffer, 0, length * 8);
57
58     auto msgs = main_rng.splitSize(64);
59
60     for (size_t i = 0; msgs.size() > i; i++)
61         this->decodeBlock(msgs[i]);
62
63     return out;
64 }
65
66 void DES::encodeBlock(BitRange current)
67 {
68     if (64 != current.size())
69         throw;
70
71     char buffer[8], key_buffer[6];
72     BitRange buf_rng (buffer, 0, 64),
73              key_rng (key_buffer, 0, 48),
74              buf_32(buffer, 0, 32);
75
76     this->ip_block->direct(current, buf_rng);
77     current.copy(buf_rng);
78     auto splits = current.split(2);
79     BitRange l = splits[0], r = splits[1];
80
81     for (size_t i = 0; 16 > i; i++)
```

```
82      {
83          this->key_block->get(i, key_rng);
84          this->f_block->apply(r, key_rng, buf_32);
85          buf_32 ^= l;
86          l.copy(r);
87          r.copy(buf_32);
88      }
89
90      this->ip_block->reverse(current, buf_rng);
91      current.copy(buf_rng);
92  }
93
94  void DES::decodeBlock(BitRange current)
95  {
96      if (64 != current.size())
97          throw;
98
99      char buffer[8], key_buffer[6];
100     BitRange buf_rng (buffer, 0, 64),
101              key_rng (key_buffer, 0, 48),
102              buf_32(buffer, 0, 32);
103
104     this->ip_block->direct(current, buf_rng);
105     current.copy(buf_rng);
106     auto splits = current.split(2);
107     BitRange l = splits[0], r = splits[1];
108
109     for (size_t i = 0; 16 > i; i++)
110     {
111         this->key_block->get(15 - i, key_rng);
112         this->f_block->apply(l, key_rng, buf_32);
113         buf_32 ^= r;
114         r.copy(l);
115         l.copy(buf_32);
116     }
117
118     this->ip_block->reverse(current, buf_rng);
119     current.copy(buf_rng);
120 }
```

Листинг 3.2 – Класс, реализующий функцию Фейстеля

```
FBlock::FBlock(std::shared_ptr<EBlock> e_block,
               std::vector<std::shared_ptr<SBlock>> s_blocks,
               std::shared_ptr<FPBlock> p_block)
    : e_block(e_block), s_blocks(s_blocks), p_block(p_block)
{
    if (nullptr == e_block || nullptr == p_block)
        throw;

    for (auto block : s_blocks)
        if (nullptr == block)
            throw;
}

void FBlock::apply(BitRange in, BitRange key, BitRange out)
{
    if (32 != in.size() || 48 != key.size()
        || 32 != out.size())
        throw;

    char buffer[6], out_buffer[4];
    BitRange buf_rng (buffer, 0, 48),
             out_rng (out_buffer, 0, 32);
    auto splits_in = buf_rng.split(8);
    auto splits_out = out_rng.split(8);

    this->e_block->direct(in, buf_rng);
    buf_rng ^= key;

    for (size_t i = 0; 8 > i; i++)
        this->s_blocks[i]->direct(splits_in[i],
                                  splits_out[i]);

    this->p_block->direct(out_rng, out);
}
```

Листинг 3.3 – Класс, реализующий получение ключа

```
KeyBlock::KeyBlock(BitRange key,
                   std::shared_ptr<PKeyBlock> p_block,

  std::vector<std::shared_ptr<ShiftKeyBlock>> shift_blocks,
```

```
4                       std::shared_ptr<HKeyBlock> h_block)
5     : p_block(p_block), shift_blocks(shift_blocks),
6       h_block(h_block)
7 {
8     if (64 != key.size())
9         throw;
10
11    if (nullptr == p_block || nullptr == h_block)
12        throw;
13
14    for (auto block : shift_blocks)
15        if (nullptr == block)
16            throw;
17
18    char buf[7];
19    BitRange inner (buf, 0, 56);
20
21    if (!this->check(key, inner))
22        throw;
23
24    this->init(inner);
25 }
26
27 void KeyBlock::get(size_t i, BitRange out)
28 {
29    if (16 <= i)
30        throw;
31
32    BitRange tmp (this->keys[i], 0, 48);
33    out.copy(tmp);
34 }
35
36 bool KeyBlock::check(BitRange key, BitRange out)
37 {
38    bool k = true;
39    char sum = 0;
40    auto iterk = key.begin(), itero = out.begin();
41
42    for (size_t i = 0; 8 > i; i++, ++iterk)
43    {
44        sum = 0;
```

```
45
46          for (size_t j = 0; 7 > j; j++, ++itero, ++iterk)
47              sum += (*itero = *iterk) ? 1 : 0;
48
49          if (sum % 2 != ((*iterk) ? 1 : 0))
50              k = false;
51      }
52
53      return k;
54  }
55
56  void KeyBlock::init(BitRange key)
57  {
58      if (56 != key.size())
59          throw;
60
61      char buffer[7], tmp[6];
62      BitRange buf_rng (buffer, 0, 56), shift_rng (tmp, 0, 28),
63               out_rng (tmp, 0, 48);
64      auto splits_buf = buf_rng.split(2);
65      BitRange c = splits_buf[0], d = splits_buf[1];
66
67      this->p_block->direct(key, c, d);
68
69      for (size_t i = 0; 16 > i; i++)
70      {
71          this->shift_blocks[i]->direct(c, shift_rng);
72          c.copy(shift_rng);
73          this->shift_blocks[i]->direct(d, shift_rng);
74          d.copy(shift_rng);
75
76          this->h_block->direct(buf_rng, out_rng);
77
78          memmove(this->keys[i], tmp, 6);
79      }
80  }
81
82  void KeyBlock::getKey(BitRange in, BitRange out)
83  {
84      if (64 != in.size() || 56 != out.size())
85          throw;
```

```
86
87      bool valid = true;
88      auto iter_in = in.begin(), iter_out = out.begin();
89
90      for (size_t i = 0; valid && 8 > i; i++, ++iter_in)
91      {
92          int sum = 0;
93
94          for (size_t j = 0;
95               valid && 7 > j;
96               j++, ++iter_out, ++iter_in)
97              sum += (*iter_out = *iter_in) ? 1 : 0;
98
99          if ((*iter_in ? 1 : 0) != sum % 2)
100             valid = false;
101     }
102
103     if (!valid)
104         throw;
105 }
```

Код остальных блоков является выборкой по таблице / массиву и не будет приведен в отчете.

3.2 Тестирование

№	Исходные данные	Ожидаемый результат	Фактический результат
1	abcdefgh ijklmnop qrstuvwx yz\0a	abcdefgh ijklmnop qrstuvwx yz\x0a\x00\x00\x00\x00\x00	abcdefgh ijklmnop qrstuvwx yz\x0a\x00\x00\x00\x00\x00
2	Пустой файл	Пустой файл	Пустой файл
3	Попытка расшифровать сообщение из теста №1 с другим ключом (во втором блоке)	abcdefgh <3 произвольных блока по 8 байт>	abcdefgh \xb7\xf9\x12\x62\xad\xec \xe5\x78\xaf\xe1\x0a\x7a \xb5\xf4\xfd\x70\xa7\xe9 \x73\x0e\xc0\x82\x8a\x08

Заключение

Была разработана программы шифрования симметричным алгоритмом DES с применением режима шифрования PCBC.

Были решены следующие задачи:

1) изучен алгоритм шифрования и применяемый режим;
2) разработан алгоритм работы программы;
3) реализовано и протестировано программное обеспечение.