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**»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Студент Романов Семен Константинович

Группа ИУ7-75Б

Предмет Защита информации

Студент		**Романов С. К.**
	подпись, дата	*фамилия, и.о.*
Преподаватель		**Чиж И. С.**
	подпись, дата	*фамилия, и.о.*

2023 г.

ВВЕДЕНИЕ

Цель лабораторной работы — реализовать программу шифрования симметричным алгоритмом DES [1] с применением PCBC [2] режима шифрования.

Задачи лабораторной работы:

1) провести анализ симметричного алгоритма шифрования DES и PCBC режима шифрования;
2) описать вышеперечисленные алгоритмы;
3) релизовать програмное обеспечение с использованием описанных алгоритмов.

1 Аналитическая часть

1.1 DES

DES — это блочный шифр, означающий, что он оперирует блоками открытого текста заданного размера (64 бита) и возвращает блоки зашифрованного текста того же размера. Таким образом, DES приводит к перестановке среди 2^{64} возможных расположений 64-х бит, каждое из которых может быть либо 0, либо 1. Каждый блок из 64 бит делится на два блока по 32 бита каждый, левая половина блока L и правая половина R.

Пример: Пусть M — обычное текстовое сообщение

$\mathtt{M} = 0123456789ABCDEF$,

где M находится в шестнадцатеричном формате (основание 16).

Переписав M в двоичном формате, мы получим 64-битный блок текста:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

$\mathtt{L} = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111$

$\mathtt{R} = 1000\ 1001\ 1010\ 1011\ 1100\ 1101\ 1110\ 1111$

Первый бит M равен «0». Последний бит равен «1».

DES работает с 64-битными блоками, используя ключи размером 56 бит. Ключи фактически хранятся как имеющие длину 64 бита, но каждый 8-й бит в ключе не используется (т.е. биты под номерами 8, 16, 24, 32, 40, 48, 56, и 64).

Пример: Пусть K — шестнадцатеричный ключ:

$\mathtt{K} = 133457799BBCDFF1$.

Это результирует в качестве двоичного ключа (устанавливая 1 = 0001, 3 = 0011 и т.д. и группируя вместе каждые восемь битов, из которых последний в каждой группе будет неиспользуемым):

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

Алгоритм DES использует следующие шаги:

1.1.1 Шаг 1: Создание ключей

64-разрядный ключ переставляется в соответствии со следующей таблицей, PC-1, приведенная как таблица 1. Поскольку первая запись в таблице равна ”57 это означает, что 57-й бит исходного ключа K становится первым битом переставленного ключа K+, 49-й бит исходного ключа становится вторым битом переставленного ключа, 4-й бит исходного ключа является последним битом переставленного ключа и т.д.

Таблица 1 – PC-1

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

Пример: Из исходного 64-разрядного ключа

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

получается 56-битную перестановку

$^{+}$ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Затем этот ключ разделяется на левую и правую половины, C0 и D0, где каждая половина содержит 28 бит.

Пример: Из переставленного ключа K^{+} мы получаем

C0 = 1111000 0110011 0010101 0101111

D0 = 0101010 1011001 1001111 0001111

Определив C0 и D0, создаются шестнадцать блоков Cn и Dn, $1 <= n <= 16$. Каждая пара блоков Cn и Dn формируется из предыдущей пары Cn-1 и Dn-1, соответственно, для n = 1, 2, ..., 16, используя следующий схеме ”сдвигов влево”предыдущего блока. Чтобы выполнить сдвиг влево, перемещается каждый бит на одно место влево, за исключением первого бита, который циклически перемещается до конца блока.

Таблица 2 – Ротация битов

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

После этого формируются ключи Kn для 1<=n<=16, применяя таблицу 3 перестановок к каждой из сцепленных пар CnDn. Каждая пара содержит 56 бит, но PC-2 использует только 48 из них.

Таблица 3 – PC-2

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

Следовательно, первый бит Kn является 14-м битом CnDn, второй бит - 17-м и так далее, заканчивая 48-м битом Kn, являющимся 32-м битом CnDn.

1.1.2 Шаг 2: Кодировка 64-битных блоков

Существует начальная перестановка IP, представленная в таблице 4 из 64 бит данных сообщения M. Это переупорядочивает биты в соответствии со следующей таблицей, где записи в таблице показывают новое расположение битов по сравнению с их первоначальным порядком. 58-й бит M становится первым битом IP, 50-й бит M становится вторым битом IP, и т.д. до 7-го бита M — последнего бита IP.

Пример: Применяя начальную перестановку к блоку текста M, приведенному ранее, получается следующее:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Здесь 58-й бит M равен ”1 который становится первым битом IP. 50-й бит M равен ”1 который становится вторым битом IP. 7-й бит M равен ”0 который становится последним битом IP.

Затем перестановочный блок IP разделяется на левую половину L_0 из 32

Таблица 4 – 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

бит и правую половину R_0 из 32 бит.

Пример: Из IP мы получаем L_0 и R_0

$L_0 = 11001100000000000110011001111111$ $R_0 = 11110000101010101111000010$

После этого выполняется 16 итераций для 1<=n<=16, используя функцию f, которая оперирует двумя блоками — блоком данных из 32 бит и ключом Kn из 48 бит — для получения блока из 32 бит. Пусть + обозначает сложение XOR, (побитовое сложение по модулю 2). Затем для n, идущих от 1 до 16, мы вычисляем:

$L_n = R_{n-1}$

$R_n = L_{n-1} + f(R_{n-1}, K_n)$

Это приводит к получению конечного блока, для n = 16, из L_{16} R_{16}. То есть, на каждой итерации мы берем правые 32 бита предыдущего результата и превращаем их в левые 32 бита текущего шага. Для правых 32 бит на текущем шаге мы выполняем XOR для левых 32 бит предыдущего шага с вычислением f .

Пример: Для n = 1 мы имеем

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010

$R_1 = L_0 + f(R_0, K_1)$

Чтобы вычислить $f(R_0, K_1)$, сначала расширяется каждый блок R_{n-1} с 32 бит до 48 бит. Это делается с помощью таблицы выбора, которая повторяет некоторые биты в R_{n-1}. Данная таблица представлена как таблица 5 и будет именоваться как E. Таким образом, $E(R_{n-1})$ имеет 32-битный входной блок и 48-битный выходной блок. Пусть E таково, что 48 бит его выходных данных, записанных в виде 8 блоков по 6 бит в каждом, получены путем выбора битов на его входных данных по порядку в соответствии со следующей таблицей:

Таблица 5 – E

2	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

Таким образом, первые три бита $E(R_{n-1})$ являются битами в позициях 32, 1 и 2 R_{n-1}, в то время как последние 2 бита $E(R_{n-1})$ являются битами в позициях 32 и 1.

Пример: Вычисляется $E(R_0)$ из R_0 следующим образом:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010 $E(R_0)$ = 011110 100001 010101 010101 011110 100001 010101 010101

Далее в вычислении функции f выполняется XOR для выходных данных $E(R_{n-1})$ с помощью ключа K_n:

$K_n + E(R_{n-1})$.

Пример: Для K_1, $E(R_0)$ имеется:

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

$E(R_0)$ = 011110 100001 010101 010101 011110 100001 010101 010101

$K_1 + E(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111.

Далее предыдущий результат, который равен 48 битам, записывается в виде:

$K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8$,

где каждая B_i представляет собой группу из шести битов.

После этого вычисляется:

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$

где $S_i(B_i)$ относится к выходным данным i-го блока S.

Если S_1 — это функция, определенная в этой таблице, а B — блок из 6 бит, то $S_1(B)$ определяется следующим образом:

Первый и последний биты B представляются как двоичное число в десятичном диапазоне от 0 до 3 (или двоичном от 00 до 11). Пусть этим числом будет i. Средние 4 бита B представляют как двоичное число в десятичном диапазоне от 0 до 15 (двоичный код от 0000 до 1111). Пусть это число равно j. В таблице находится число в i-й строке и j-м столбце. Это число в диапазоне от 0 до 15 и однозначно представлено 4-битным блоком. Этот блок является выходом $S_1(B)$ из S_1 для входного блока B. Например, для входного блока $B = 011011$ первый бит равен «0», а последний бит «1» дает 01 в качестве строки. Это строка 1. Средние четыре бита - это «1101». Это двоичный эквивалент десятичного числа 13, поэтому столбец имеет номер 13. В строке 1 и столбце 13 помещено число 5, что и определяет выходные данные; 5 — двоичное значение которого равно 0101 — следовательно, $S_1(011011) = 0101$. Таблицы для определения $S_1 ... S_8$ показаны на рисунке 1:

Таблица 3. Преобразования S_i, 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	S_1
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	S_2
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	S_3
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	S_4
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	S_5
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	S_6
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	S_7
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	S_8
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	

Рисунок 1 – Таблицы $S_1 \ldots S_8$

Заключительным этапом вычисления функции f является выполнение перестановки P выходных данных для получения конечного значения функции f:

$f = P(S_1(B_1)S_2(B_2)...S_8(B_8))$

Перестановка P определена таблице 6. P выдает 32-разрядный выходной сигнал из 32-разрядного входного сигнала путем перестановки битов входного блока.

Таблица 6 – 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

Пример: Из выходных данных восьми блоков S:

$S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

мы получаем

f = 0010 0011 0100 1010 1010 1001 1011 1011

$R_1 = L_0 + f(R_0, K_1)$

= 1100 1100 0000 0000 1100 1100 1111 1111

+ 0010 0011 0100 1010 1010 1001 1011 1011

= 1110 1111 0100 1010 0110 0101 0100 0100

В следующем раунде у нас будет $L_2 = R_1$, который является блоком, который мы только что вычислили, а затем мы должны вычислить $R_2 = L_1 + f(R_1, K_2)$, и так далее в течение 16 раундов. В конце шестнадцатого раунда у

нас есть блоки $L_{16}R_{16}$. Затем мы меняем порядок расположения двух блоков на 64-битный блок

$R_{16}L_{16}$

и примените окончательную перестановку IP-1, определененную в таблице 7:

Таблица 7 – 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

То есть выходные данные алгоритма содержат 40-ой бит блока предварительного вывода в качестве первого бита, бит 8 в качестве второго бита и так далее, пока бит 25 блока предварительного вывода не станет последним битом выходных данных.

Пример: Если мы обработаем все 16 блоков, используя метод, определенный ранее, мы получим в 16-ой итерации,

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100

R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

Порядок расположения этих двух блоков меняются на противоположный и применяется окончательную перестановку к

$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

который в шестнадцатеричном формате равен `85E813540F0AB405`.

Таким образом $DES(0123456789ABCDEF) = 85E813540F0AB405$.

Дешифрование - это просто обратная операция шифрования, выполняющая те же шаги, что и описанные выше, но в обратном порядке, в котором применяются подразделы.

1.2 PCBC

Недостатки режима `CBC` привели к созданию усовершенствованного режима распространяющегося сцепления блоков шифра (`Propagating Cipher Block Chaining`,). Естественно, этот режим похож на `CBC` за исключением того, что предыдущий блок открытого текста и предыдущий блок шифротекста подвергается операции XOR с текущим блоком открытого текста перед шифрованием или после него.

$c_i = E_k\left(m_i \oplus m_{i-1} \oplus c_{i-1}\right)$

Соответственно расшифрование:

$m_i = D_k(c_i) \oplus c_{i-1} \oplus m_{i-1}$

где

$m_0 \oplus c_0$ — вектор инициализации.

Данный режим шифрования не является федеральным или международным стандартом. Режим — вариант режима , обладающий специфическим свойством — ошибка шифротекста приводит к неправильному дешифрированию всех последующих блоков. Это соответственно означает, что проверка стандартного блока в конце сообщения обеспечивает целостность всего сообщения.

2 Конструкторская часть

2.1 Разработка алгоритма

На рисунках 2 приведена схема работы PCBC шифрования.

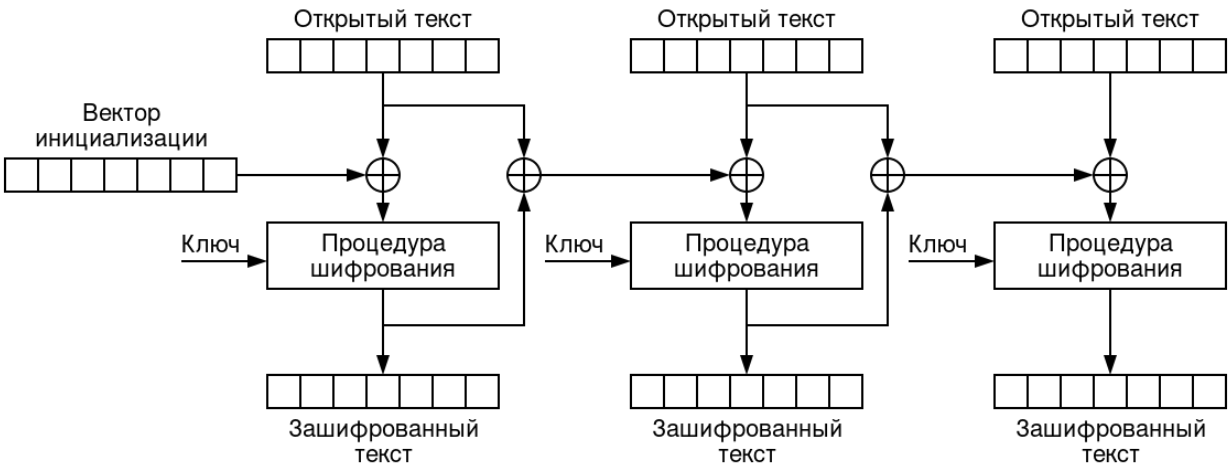

Рисунок 2 – Схема работы PCBC шифрования

На рисунке 3 приведена схема работы алгоритма DES.

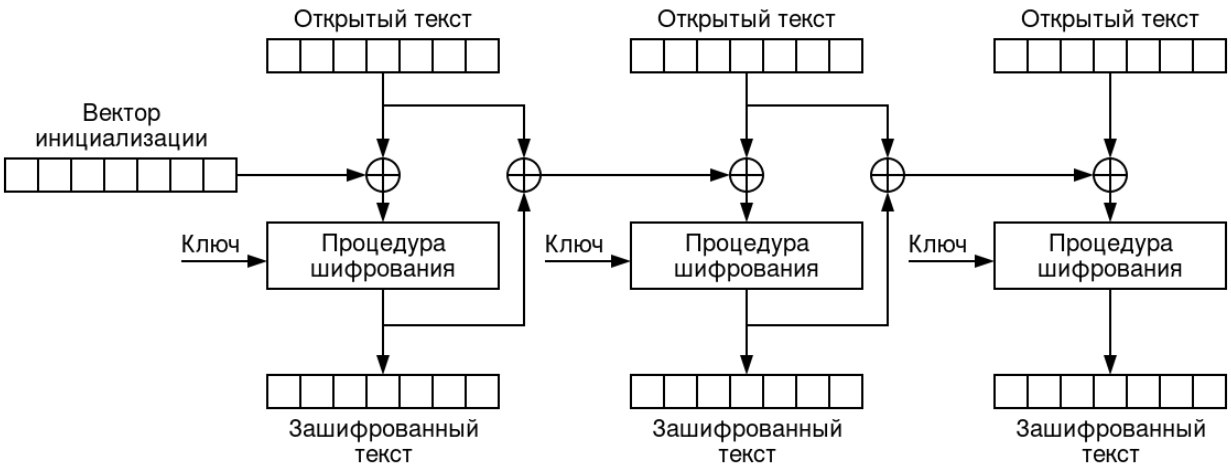

Рисунок 3 – Схема работы алгоритма DES

3 Технологическая часть

3.1 Средства реализации

Для реализации ПО был выбран язык C++ [3]. В данном языке есть все требующиеся инструменты для данной лабораторной работы. В качестве среды разработки была выбрана среда NeoVim[4].

3.2 Реализация алгоритма

Листинг 1 – Реализация алгоритма DES.

```
std::string DES::run(const std::string& DATA)
{
    if(!keyset)
    {
        throw std::runtime_error("Error: Key has not been set.");
    }

    if(DATA.size() != 8)
    {
        throw std::runtime_error("Error: Data must be 64 bits in length.");
    }

    std::string data = "", temp = "";
    for(uint8_t x = 0; x < 8; x++)
    {
        data += makebin(static_cast<uint8_t>(DATA[x]), 8);
    }
    // IP
    for(uint8_t x = 0; x < 64; x++)
    {
        temp += data[IP[x] - 1];
    }
    data = temp;
    for(uint8_t x = 0; x < 16; x++)
    {
        // split left and right and duplicate right
        std::string left = data.substr(0, 32);
        std::string right = data.substr(32, 32);
        std::string old_right = right;
```

Листинг 2 – Реализация алгоритма DES, часть 2.

```
        // expand right side
        uint64_t t = 0;
        temp = "";
        for(uint8_t y = 0; y < 48; y++)
        {
            temp += right[EX[y] - 1];
        }
        t = toint(temp, 2);

        // expanded_right xor key
        right = makebin(t ^ keys[x], 48);

        // split right into 8 parts
        std::string RIGHT[8];
        for(uint8_t y = 0; y < 8; y++)
        {
            RIGHT[y] = right.substr(6 * y, 6);
        }
        // use sboxes
        temp = "";
        for(uint8_t y = 0; y < 8; y++)
        {
            std::string s = "";
            s += RIGHT[y][0];
            s += RIGHT[y][5];
            temp += makebin(S_BOX[y][toint(s, 2)][toint(RIGHT[y].substr(1, 4),
                2)], 4);
        }

        // permutate
        right = "";
        for(uint8_t y = 0; y < 32; y++)
        {
            right += temp[P[y] - 1];
        }

        // right xor left and combine with old right
        data = old_right + makebin(toint(left, 2) ^ toint(right, 2), 32);
```

Листинг 3 – Реализация алгоритма DES, часть 3.

```
    }
    // reverse last switch
    data = data.substr(32, 32) + data.substr(0, 32);

    // IP^-1
    uint64_t out = 0;
    for(uint8_t x = 0; x < 64; x++)
    {
        out += static_cast<uint64_t>(data[INVIP[x] - 1] == '1') << (63 - x);
    }
    return unhexlify(makehex(out, 16));
}
```

Листинг 4 – Реализация алгоритма шифрования PCBC.

```
std::string PCBC::encrypt(const std::string& data)
{
    std::string temp = pkcs5(data, blocksize);
    std::string out = "";
    std::string IV = const_IV;
    for(std::string::size_type x = 0; x < temp.size(); x += blocksize)
    {
        const std::string block = temp.substr(x, blocksize);
        out += algo->encrypt(xor_strings(block, IV));
        IV = xor_strings(out.substr(out.size() - blocksize, blocksize), block);
    }
    return out;
}
{
    std::string out = "";
    std::string IV = const_IV;
    for(std::string::size_type x = 0; x < data.size(); x += blocksize)
    {
        const std::string block = data.substr(x, blocksize);
        out += xor_strings(algo->decrypt(block), IV);
        IV = xor_strings(out.substr(out.size() - blocksize, blocksize), block);
    }
    return remove_pkcs5(out);
}
```

3.3 Тестовые данные

В таблице 8 приведены тесты, описанные в листинге 5 для алгоритма DES. Применена методология черного ящика. Тесты пройдены *успешно*.

Листинг 5 – Реализация функциональных тестов.

```
TEST(DES, set1)
{
    sym_test<DES>(DES_NESSIE_SET_1);
}

TEST(DES, set2)
{
    sym_test<DES>(DES_NESSIE_SET_2);
}

TEST(DES, set3)
{
    sym_test<DES>(DES_NESSIE_SET_3);
}
```

Таблица 8 – Функциональные тесты

Входная строка	Ключ	Выходная строка
0000000000000000	8000000000000000	95A8D72813DAA94D
EA024714AD5C4D84	2BD6459F82C5B300	126EFE8ED312190A
F0F0F0F0F0F0F0F0	F0F0F0F0F0F0F0F0	2A2891F65BB8173C
0011223344556677	0001020304050607	3EF0A891CF8ED990

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе:

1) проведен анализ симметричного алгоритма шифрования DES и PCBC режима шифрования;
2) описаны вышеперечисленные алгоритмы;
3) релизовано програмное обеспечение с использованием описанных алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Список литературы

1. The DES Algorithm Illustrated. https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.html. дата обращения: 14.10.2023.

2. What are the Counter and PCBC Modes? http://altlasten.lutz.donnerhacke.de/mitarb/lutz/security/cryptfaq/q84.html. дата обращения: 14.10.2023.

3. Язык программирования C++. https://learn.microsoft.com/en-us/cpp/cpp/cpp-language-reference?view=msvc-170. дата обращения: 14.10.2023.

4. Neovim. https://neovim.io/. дата обращения: 14.10.2023.