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»

Лабораторная работа по дисциплине «Защита информации»

Тема Алгоритм шифрования AES и режимы шифрования

Студент Светличная А.А.

Группа ИУ7-73Б

Преподаватель Чиж И. С.

Москва — 2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, где информационная безопасность и конфиденциальность данных становятся все более актуальными вопросами, использование надежных методов шифрования становится обязательным условием для защиты ценных информационных ресурсов. В этом контексте, Advanced Encryption Standard, или AES, занимает выдающееся место в арсенале криптографических инструментов. Этот симметричный алгоритм шифрования, утвержденный Национальным институтом стандартов и технологий США (NIST) в 2001 году, стал неотъемлемой частью мировой криптографической практики.

Цель: разработка программной реализации шифровального алгоритма AES в режиме работы по варианту.

Задачи:

- исследование исторических аспектов данного алгоритма;
- анализ алгоритма AES;
- программная реализация данного алгоритма.

1 Аналитическая часть

Историческая справка

AES (Advanced Encryption Standard) — это симметричный алгоритм шифрования, который был утвержден в качестве стандарта национальным институтом стандартов и технологий США (NIST) в 2001 году. AES стал приоритетным стандартом шифрования для правительственных и коммерческих организаций по всему миру и заменил более ранний стандарт, известный как DES (Data Encryption Standard).

История AES началась в 1997 году, когда NIST объявил конкурс на создание нового стандарта шифрования, который должен был обеспечивать высокую степень безопасности и эффективности. Конкурс привлек множество кандидатов, и в конечном итоге, пять финалистов были выбраны для более подробного анализа: MARS, RC6, Rijndael, Serpent и Twofish.

В 2000 году NIST провел серию открытых обсуждений и анализов, в результате которых Rijndael, предложенный бельгийскими криптографами Винсентом Рижем и Жан-Жаком Квиске, был выбран в качестве победителя и признан новым стандартом AES. Его выбор основывался на его выдающейся стойкости к атакам, скорости и эффективности, а также на простоте реализации.

AES использует симметричный ключевой обмен, что означает, что один и тот же ключ используется как для шифрования, так и для дешифрования данных. Он поддерживает разные длины ключей (128 бит, 192 бита и 256 бит), что позволяет выбирать уровень безопасности в зависимости от конкретных потребностей.

Этот стандарт оказался успешным и широко применяется в различных областях, включая информационную безопасность, финансовые услуги, облачные вычисления и многое другое. AES считается одним из наиболее надежных алгоритмов шифрования и остается важной частью современных технологий безопасности данных.

2 Конструкторская часть

2.1 Шифровальный алгоритм DES

Процесс шифрования состоит из следующих этапов:

1. Расширение ключа — KeyExpansion.
2. Начальный раунд — AddRoundKey с основным ключом.
3. Раунда шифрования:
   - SubBytes;
   - ShiftRows;
   - MixColumns;
   - AddRoundKey;
4. Финальный раунд:
   - SubBytes;
   - ShiftRows;
   - AddRoundKey;

На рисунках 2.1, 2.2, 2.3, 2.4 показаны схемы для одного «шага» операций, выполняющихся в раундах шифрования.

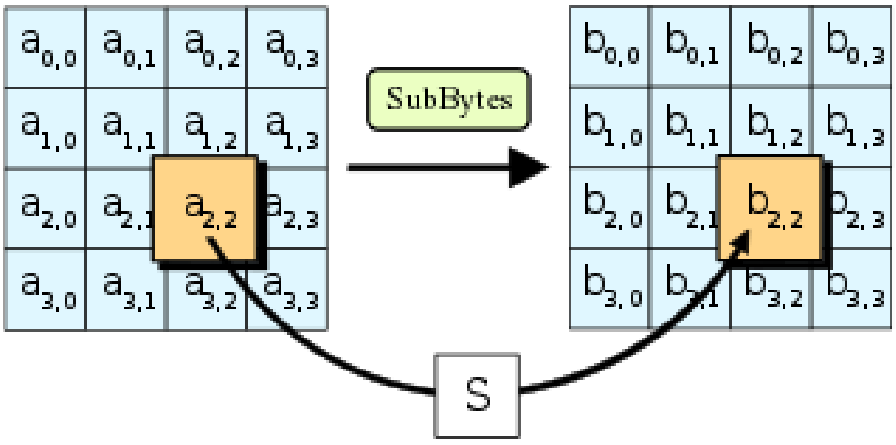

Рисунок 2.1 – SubBytes

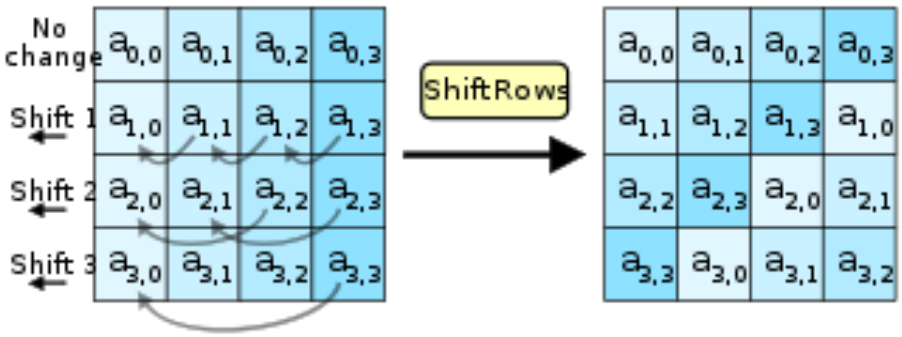

Рисунок 2.2 – ShiftRows

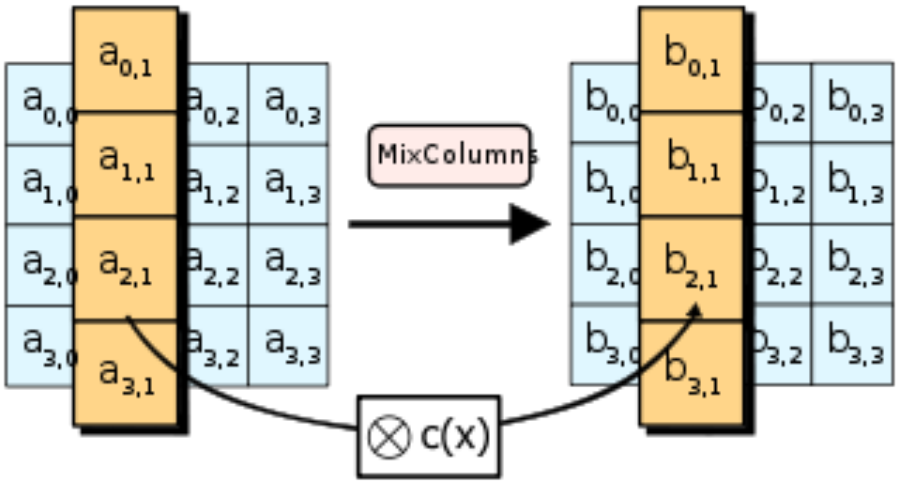

Рисунок 2.3 – MixColumns

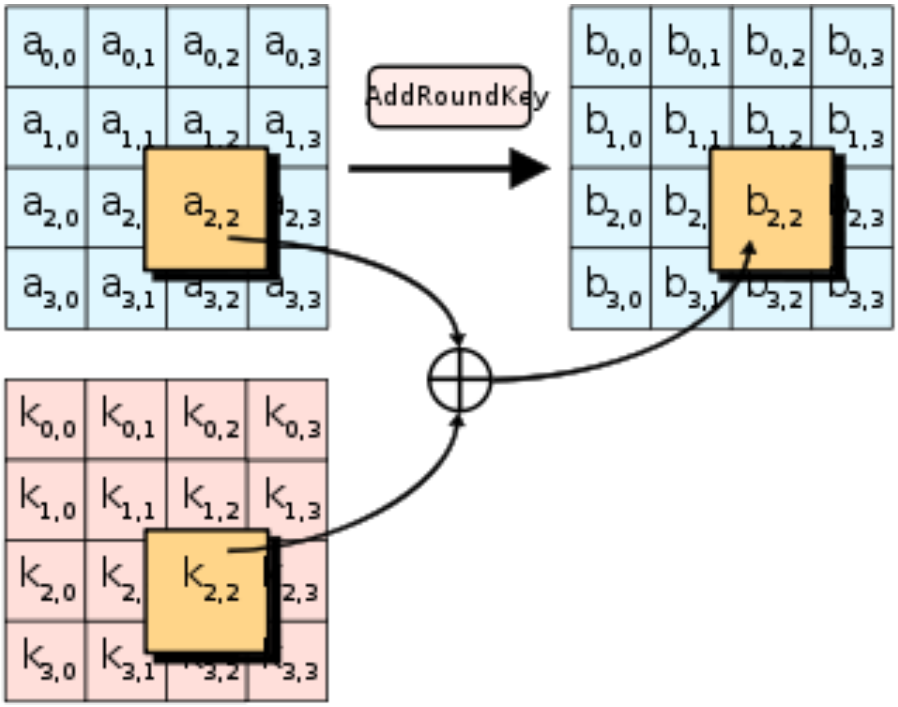

Рисунок 2.4 – AddRoundKey

2.2 Режим работы

Чтобы воспользоваться алгоритмом AES для решения разнообразных криптографических задач, разработаны рабочие режимы:

1. Электронная кодовая книга (ECB — Electronic Code Book).
2. Сцепление блоков шифра (CBC — Cipher Block Chaining).
3. Распространяющееся сцепление блоков шифра (PCBC — Propagating Cipher Block Chaining).
4. Обратная связь по шифртексту (CFB — Cipher Feed Back).
5. Обратная связь по выходу (OFB — Output Feed Back).

На рисунке 2.5 показана схема шифрования для режима PCBC.

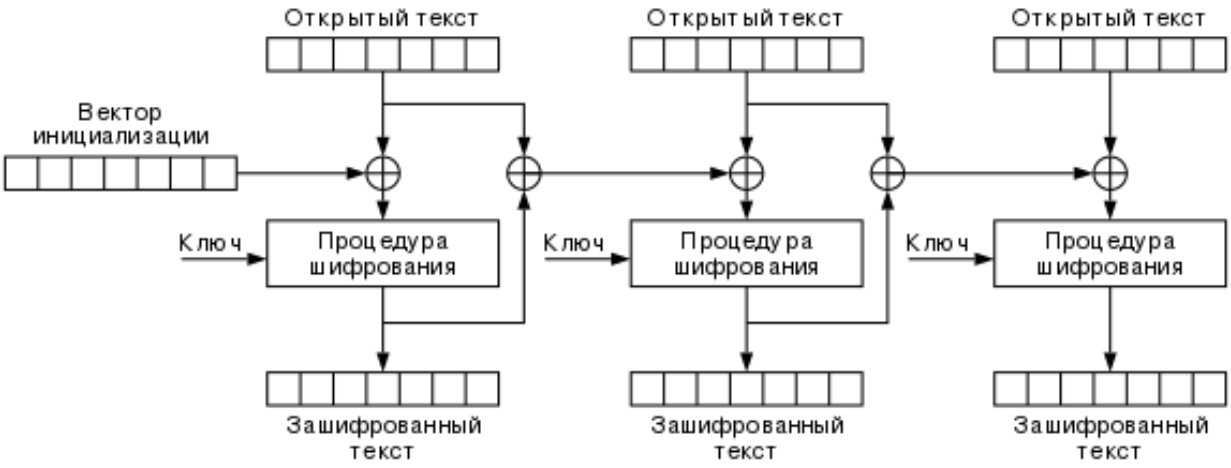

Рисунок 2.5 – Режи шифрования PCBC

3 Технологическая часть

3.1 Реализация алгоритма

Листинг 3.1 – Реализация части работы с ключами алгоритма AES

```
1  void expandKey()
2  {
3      int currentSize = 0;
4      int rconIteration = 1;
5      unsigned char t[4] = { 0 };
6
7      for (int i = 0; i < keySize(); i++)
8          expandedKey[i] = key[i];
9      currentSize += keySize();
10
11     while (currentSize < expandedKeySize())
12     {
13         for (int i = 0; i < 4; i++)
14             t[i] = expandedKey[(currentSize - 4) + i];
15
16         if (currentSize % keySize() == 0)
17             core(t, rconIteration++);
18
19         if (keySize() == 32 && ((currentSize % keySize()) ==16))
20             for (int i = 0; i < 4; i++)
21                 t[i] = Sbox[t[i]];
22
23         for (int i = 0; i < 4; i++)
24         {
25             expandedKey[currentSize] = expandedKey[currentSize -
                   ↪ keySize()] ^ t[i];
26             currentSize++;
27         }
28     }
29 }
```

```
30 void core(unsigned char* word, int iteration)
31 {
32     unsigned char tmp = word[0];
33 
34     for (int i = 0; i < 3; i++)
35         word[i] = word[i + 1];
36     word[3] = tmp;
37 
38     for (int i = 0; i < 4; ++i)
39         word[i] = Sbox[word[i]];
40 
41     word[0] = word[0] ^ Rcon[iteration];
42 }
```

Листинг 3.2 – Реализация шифрования/расшифровки алгоритма AES в режиме PCBC

```
int main(int argc, char* argv[])
{
    ...
    int i = 0;
    while (i++ < numBlock)
    {
        readText(file);

        for (int i = 0; i < 16; i++)
            IVxorText[i] = text[i] ^ IV[i];

        aes();
        aes_inv();

        for (int i = 0; i < 16; i++)
            decrypted[i] = decrypted[i] ^ IV[i];

        print();
        printFile();

        for (int i = 0; i < 16; i++)
            IV[i] = text[i] ^ encrypted[i];
    }
    ...
}

void subBytes(unsigned char* state)
{
    for (int i = 0; i < 16; i++)
        state[i] = Sbox[state[i]];
}

```

```
33 void shiftRows(unsigned char* state)
34 {
35     for (int i = 0; i < 4; i++)
36     {
37         for (int j = 0; j < i; j++)
38         {
39             int tmp = state[i];
40             for (int k = 0; k < 3; k++)
41                 state[k * 4 + i] = state[(k + 1) * 4 + i];
42             state[12 + i] = tmp;
43         }
44     }
45 }
46
47 void mixColumns(unsigned char* state)
48 {
49     unsigned char column[4];
50
51     for (int i = 0; i < 4; i++)
52     {
53         for (int j = 0; j < 4; j++)
54             column[j] = state[i * 4 + j];
55
56         mixColumn(column);
57
58         for (int j = 0; j < 4; j++)
59             state[i * 4 + j] = column[j];
60     }
61 }
62
63 void mixColumn(unsigned char* column)
64 {
65     unsigned char tmp[4];
66
```

```
67      for (int i = 0; i < 4; i++)
68          tmp[i] = column[i];
69
70      column[0] =
71                  galois_multiplication(tmp[0], 2) ^
72                  galois_multiplication(tmp[1], 3) ^
73                  galois_multiplication(tmp[2], 1) ^
74                  galois_multiplication(tmp[3], 1);
75
76      column[1] =
77                  galois_multiplication(tmp[0], 1) ^
78                  galois_multiplication(tmp[1], 2) ^
79                  galois_multiplication(tmp[2], 3) ^
80                  galois_multiplication(tmp[3], 1);
81
82      column[2] =
83                  galois_multiplication(tmp[0], 1) ^
84                  galois_multiplication(tmp[1], 1) ^
85                  galois_multiplication(tmp[2], 2) ^
86                  galois_multiplication(tmp[3], 3);
87
88          column[3] =
89                  galois_multiplication(tmp[0], 3) ^
90                  galois_multiplication(tmp[1], 1) ^
91                  galois_multiplication(tmp[2], 1) ^
92                  galois_multiplication(tmp[3], 2);
93  }
94
95  unsigned char galois_multiplication(unsigned char a, unsigned
       ↪ char b)
96  {
97      unsigned char p = 0;
98      unsigned char counter;
99      unsigned char hi_bit_set;
```

```
100
101     for (counter = 0; counter < 8; counter++)
102     {
103         if ((b & 1) == 1)
104             p ^= a;
105         hi_bit_set = (a & 0x80);
106         a <<= 1;
107         if (hi_bit_set == 0x80)
108             a ^= 0x1b;
109         b >>= 1;
110     }
111     return p;
112 }
113
114 void addRoundKey(unsigned char* state, unsigned char* roundKey)
115 {
116     for (int i = 0; i < 16; i++)
117         state[i] = state[i] ^ roundKey[i];
118 }
119
120 void aes_round(unsigned char* state, unsigned char* roundKey)
121 {
122     subBytes(state);
123     shiftRows(state);
124     mixColumns(state);
125     addRoundKey(state, roundKey);
126 }
127
128 void aes()
129 {
130     unsigned char state[16];
131
132     for (int i = 0; i < 16; i++)
133         state[i] = IVxorText[i];
```

```
134
135     addRoundKey(state, expandedKey);
136
137     for (int i = 1; i < numRounds(); i++)
138         aes_round(state, expandedKey + 16 * i);
139
140     subBytes(state);
141     shiftRows(state);
142     addRoundKey(state, expandedKey + 16 * numRounds());
143
144     for (int i = 0; i < 16; i++)
145         encrypted[i] = state[i];
146 }
147
148 void invSubBytes(unsigned char* state)
149 {
150     for (int i = 0; i < 16; i++)
151         state[i] = RSbox[state[i]];
152 }
153
154 void invShiftRows(unsigned char* state)
155 {
156     for (int i = 0; i < 4; i++)
157     {
158         for (int j = 0; j < i; j++)
159         {
160             int tmp = state[12 + i];
161             for (int k = 3; k > 0; k--)
162                 state[k * 4 + i] = state[(k - 1) * 4 + i];
163             state[i] = tmp;
164         }
165     }
166 }
167
```

```
168 void invMixColumns(unsigned char* state)
169 {
170     unsigned char column[4];
171 
172     for (int i = 0; i < 4; i++)
173     {
174         for (int j = 0; j < 4; j++)
175             column[j] = state[i * 4 + j];
176 
177         invMixColumn(column);
178 
179         for (int j = 0; j < 4; j++)
180             state[i * 4 + j] = column[j];
181     }
182 }
183 
184 void invMixColumn(unsigned char* column)
185 {
186     unsigned char tmp[4];
187 
188     for (int i = 0; i < 4; i++)
189         tmp[i] = column[i];
190 
191     column[0] =
192                 galois_multiplication(tmp[0], 14) ^
193                 galois_multiplication(tmp[1], 11) ^
194                 galois_multiplication(tmp[2], 13) ^
195                 galois_multiplication(tmp[3], 9);
196 
197     column[1] =
198                 galois_multiplication(tmp[0], 9) ^
199                 galois_multiplication(tmp[1], 14) ^
200                 galois_multiplication(tmp[2], 11) ^
201                 galois_multiplication(tmp[3], 13);
```

```
202
203     column[2] =
204                 galois_multiplication(tmp[0], 13) ^
205                 galois_multiplication(tmp[1], 9) ^
206                 galois_multiplication(tmp[2], 14) ^
207                 galois_multiplication(tmp[3], 11);
208
209     column[3] =
210                 galois_multiplication(tmp[0], 11) ^
211                 galois_multiplication(tmp[1], 13) ^
212                 galois_multiplication(tmp[2], 9) ^
213                 galois_multiplication(tmp[3], 14);
214 }
215
216 void aes_invRound(unsigned char* state, unsigned char* roundKey
    ↪ )
217 {
218     invShiftRows(state);
219     invSubBytes(state);
220     addRoundKey(state, roundKey);
221     invMixColumns(state);
222 }
223
224 void aes_inv()
225 {
226     unsigned char state[16];
227
228     for (int i = 0; i < 16; i++)
229         state[i] = encrypted[i];
230
231     addRoundKey(state, expandedKey + 16 * numRounds());
232
233     for (int i = numRounds() - 1; i > 0; i--)
234         aes_invRound(state, expandedKey + 16 * i);
```

```
235
236     invShiftRows(state);
237     invSubBytes(state);
238     addRoundKey(state, expandedKey);
239
240     for (int i = 0; i < 16; i++)
241         decrypted[i] = state[i];
242 }
```

3.2 Тестирование

Негативные:

1. Строка запуска: ./app.exe
   Код возврата: 1
   Описание: не задано имя входного файла

2. Строка запуска: ./app.exe noexist.txt
   Код возврата: 2
   Описание: не существует входной файл

Позитивные:

1. Строка запуска: ./app.exe input.txt

```
1     Block 1:
2
3     Cipher Key (HEX format):
4     54 68 61 74 73 20 6d 79 20 4b 75 6e 67 20 46 75
5
6     Expanded Key (HEX format):
7     54 68 61 74 73 20 6d 79 20 4B 75 6e 67 20 46 75
8     e2 32 fc f1 91 12 91 88 b1 59 e4 e6 d6 79 a2 93
9     56 08 20 07 c7 1a b1 8f 76 43 55 69 a0 3a f7 fa
10    d2 60 0d e7 15 7a bc 68 63 39 e9 01 c3 03 1e fb
11    a1 12 02 c9 b4 68 be a1 d7 51 57 a0 14 52 49 5b
12    b1 29 3b 33 05 41 85 92 d2 10 d2 32 c6 42 9b 69
```

```
    bd 3d c2 87 b8 7c 47 15 6a 6c 95 27 ac 2e 0e 4e
    cc 96 ed 16 74 ea aa 03 1e 86 3f 24 b2 a8 31 6a
    8e 51 ef 21 fa bb 45 22 e4 3d 7a 06 56 95 4b 6c
    bf e2 bf 90 45 59 fa b2 a1 64 80 b4 f7 f1 cb d8
    28 fd de f8 6d a4 24 4a cc c0 a4 fe 3b 31 6f 26

    Initialization Vector (HEX format):
    61 73 66 6a 63 6d 78 20 63 68 6b 6d 73 6b 66 66

    Plaintext (HEX format):
    54 77 6f 20 4f 6e 65 20 4e 69 6e 65 20 54 77 6f

    Encrypted text (HEX format):
    f1 8a 66 07 ad 30 35 f6 ea 8e a5 86 f3 d3 30 5f

    Decrypted text (HEX format):
    54 77 6f 20 4f 6e 65 20 4e 69 6e 65 20 54 77 6f
```

Код возврата: 0

Описание: сообщение размером более 128 бит

2. Строка запуска: ./app.exe input.txt

```
    Block 1:

    Cipher Key (HEX format):
    54 68 61 74 73 20 6d 79 20 4b 75 6e 67 20 46 75

    Expanded Key (HEX format):
    54 68 61 74 73 20 6d 79 20 4b 75 6e 67 20 46 75
    e2 32 fc f1 91 12 91 88 b1 59 e4 e6 d6 79 a2 93
    56 08 20 07 c7 1a b1 8f 76 43 55 69 a0 3a f7 fa
    d2 60 0d e7 15 7a bc 68 63 39 e9 01 c3 03 1e fb
    a1 12 02 c9 b4 68 be a1 d7 51 57 a0 14 52 49 5b
    b1 29 3b 33 05 41 85 92 d2 10 d2 32 c6 42 9b 69
```

```
    bd 3d c2 87 b8 7c 47 15 6a 6c 95 27 ac 2e 0e 4e
    cc 96 ed 16 74 ea aa 03 1e 86 3f 24 b2 a8 31 6a
    8e 51 ef 21 fa bb 45 22 e4 3d 7a 06 56 95 4b 6c
    bf e2 bf 90 45 59 fa b2 a1 64 80 b4 f7 f1 cb d8
    28 fd de f8 6d a4 24 4a cc c0 a4 fe 3b 31 6f 26

    Initialization Vector (HEX format):
    61 73 66 6a 63 6d 78 20 63 68 6b 6d 73 6b 66 66

    Plaintext (HEX format):
    54 77 6f 20 4f 6e 65 20 4e 69 6e 65 20 54 77 6f

    Encrypted text (HEX format):
    f1 8a 66 07 ad 30 35 f6 ea 8e a5 86 f3 d3 30 5f

    Decrypted text (HEX format):
    54 77 6f 20 4f 6e 65 20 4e 69 6e 65 20 54 77 6f

    Block 2:

    Cipher Key (HEX format):
    54 68 61 74 73 20 6d 79 20 4b 75 6e 67 20 46 75

    Expanded Key (HEX format):
    54 68 61 74 73 20 6d 79 20 4b 75 6e 67 20 46 75
    e2 32 fc f1 91 12 91 88 b1 59 e4 e6 d6 79 a2 93
    56 08 20 07 c7 1a b1 8f 76 43 55 69 a0 3a f7 fa
    d2 60 0d e7 15 7a bc 68 63 39 e9 01 c3 03 1e fb
    a1 12 02 c9 b4 68 be a1 d7 51 57 a0 14 52 49 5b
    b1 29 3b 33 05 41 85 92 d2 10 d2 32 c6 42 9b 69
    bd 3d c2 87 b8 7c 47 15 6a 6c 95 27 ac 2e 0e 4e
    cc 96 ed 16 74 ea aa 03 1e 86 3f 24 b2 a8 31 6a
    8e 51 ef 21 fa bb 45 22 e4 3d 7a 06 56 95 4b 6c
    bf e2 bf 90 45 59 fa b2 a1 64 80 b4 f7 f1 cb d8
```

```
    28 fd de f8 6d a4 24 4a cc c0 a4 fe 3b 31 6f 26

    Initialization Vector (HEX format):
    a5 fd 09 27 e2 5e 50 d6 a4 e7 cb e3 d3 87 47 30

    Plaintext (HEX format):
    66 67 66 64 68 72 74 62 66 00 00 00 00 00 00 00

    Encrypted text (HEX format):
    7a f7 89 49 9b 48 e0 08 3d 56 f9 3a b4 f5 55 2f

    Decrypted text (HEX format):
    66 67 66 64 68 72 74 62 66 00 00 00 00 00 00 00
```

Код возврата: 0

Описание: сообщение размером 128 бит

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы была достигнута поставленная **цель**: разработана программная реализации шифровального алгоритма AES в режиме работы по варианту.

Все **задачи** лабораторной работы выполнены:

- исследованы исторические аспекты данного алгоритма;
- проведен анализ алгоритма AES;
- программно реализован данный алгоритм.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ