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 №3

по курсу «Защита информации»

на тему: «AES»

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 AES с применением режима шифрования CFB (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

AES (Rijndael) — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит). Алгоритм шифрования базируется на применении SP-сетей, что представляют из себя последовательность применения блок замены (S) и перемешивания (P) к блоку открытого для получения блока шифр текста. Основной вычислений являются вычисления в конечном поле Галуа $GF(2^8)$.

Изначально блок сообщения преобразуется для в матрицу, элементы которой располагается по столбцам

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

Далее преобразования производятся по следующей схеме.

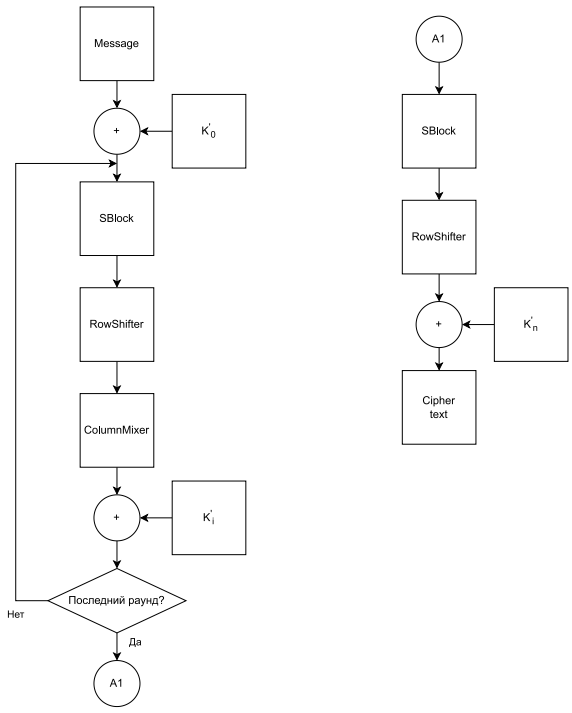

Рисунок 1.1 – Алгоритм работы AES

1.1.1 Получение раундовых ключей

Раундовые ключи K'_0–K'_n получаются при помощи расширения. Пример расчета для 128 битного ключа приведен на рисунке.

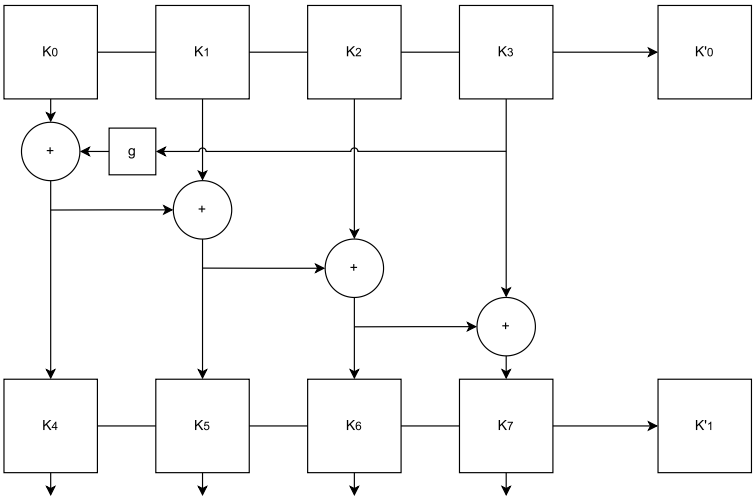

Рисунок 1.2 – Алгоритм получения ключа

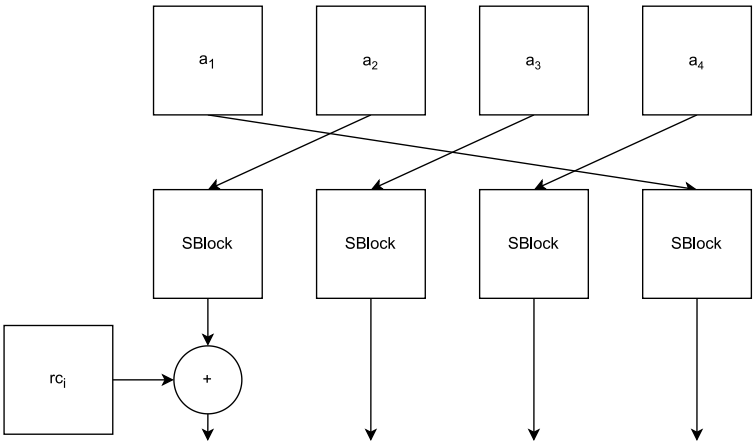

Рисунок 1.3 – Алгоритм функции g

K_1–K_4 — байты исходного ключа, rc – раундовое значение из поля Галуа $\mathrm{GF}(2^8)$, определяемое следующим соотношением $rc_i = 2 \cdot rc_{i-1}$.

1.1.2 Блока замены

Входной байт интерпретируется как многочлен поля Галуа, где значение соответствующего разряда — значение коэффициента при соответствующей степени многочлена. Последовательность действий:

1) нахождение обратного элемента;
2) умножение вектора на постоянную матрицу;
3) сумма с постоянным смещением $0x63$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix}$$

Данные значение могут быть вычислены заранее для всех 255 элементов поля и записаны в соответствующую таблицу.

Для обратного преобразования необходимо обратная операция, которая определяется следующим образом.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{bmatrix}$$

1.1.3 Блоки перестановки

Преобразование строк происходит левым циклическим сдвигом байт матрицы сообщение в зависимости от номера строки.

Перестановка столбцов происходит умножением вектора-столбца на многочлен $\begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \end{bmatrix}$. Обратное преобразование — умножением на многочлен $\begin{bmatrix} 14 \\ 9 \\ 13 \\ 11 \end{bmatrix}$.

1.2 Режим шифрования CFB

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

В данном режиме для шифрования следующего блока открытого текста происходит сложение по модулю 2 с зашифрованным результатом шифрования предыдущего блока.

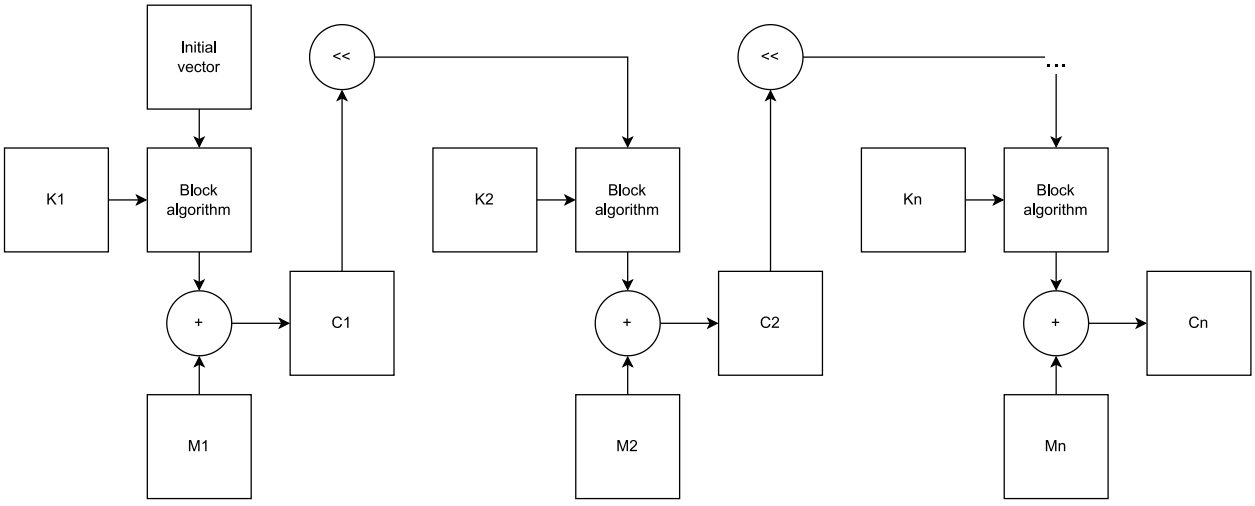

Рисунок 1.4 – Схема режима шифрования CFB

В результате, ошибка при передаче зашифрованного сообщения отразится только при расшифровке поврежденного и следующего за ним блоков.

2 Конструкторская часть

2.1 Схема алгоритма работы

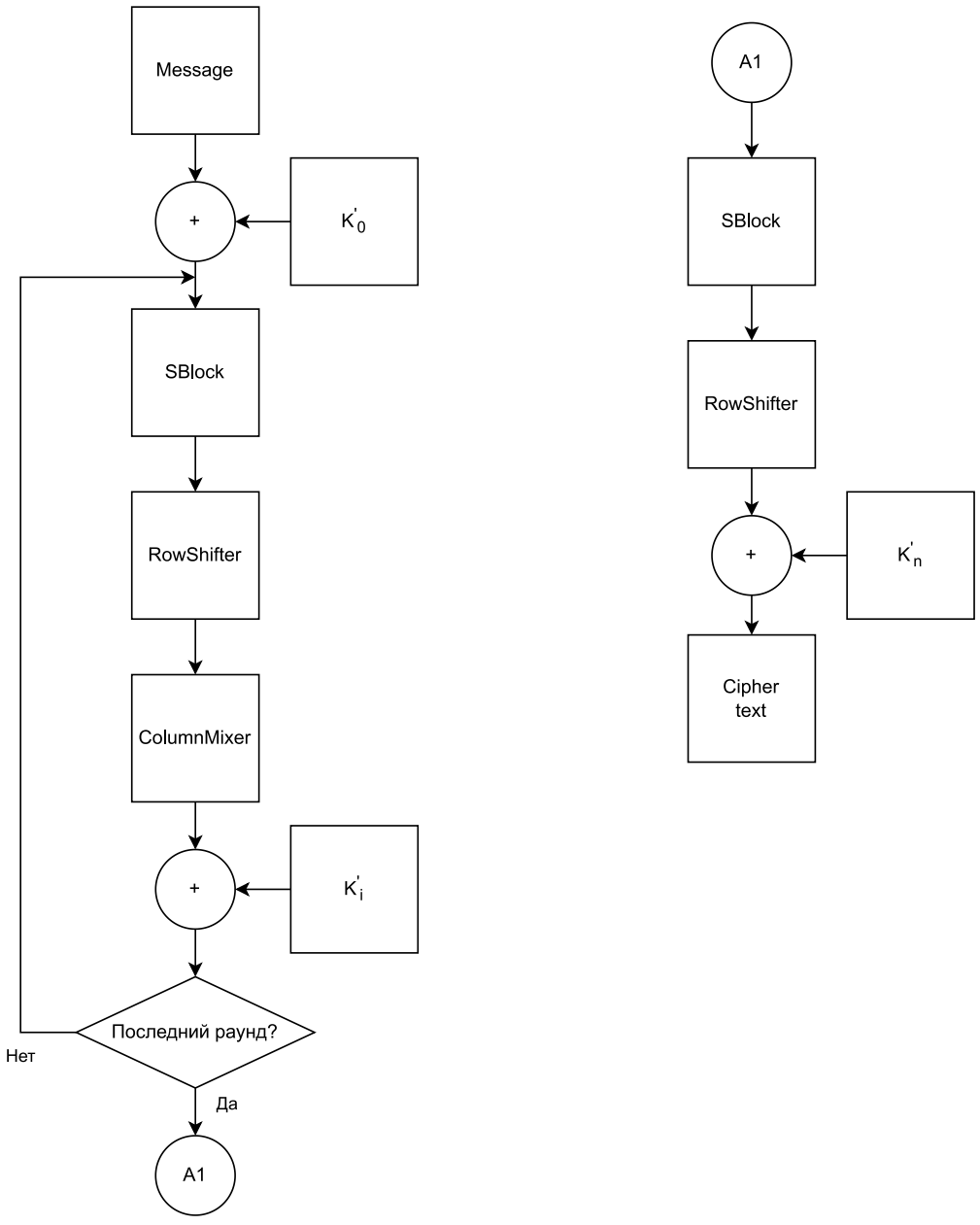

Рисунок 2.1 – Алгоритм работы AES

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_vector",
    "aes": [
        {"key": "./config/keys/key1", "offset": 1},
        {"key": "./config/keys/key2", "offset": 2},
        {"key": "./config/keys/key3", "offset": 1}
    ]
}
```

Соответсвтующие концигурационный файлы состоят из последовательности байт: 16 для начального вектора; 16, 24 или 32 для ключа.

Листинг 3.1 – Класс, реализующий алгоритм AES

```
1  void AES::encodeBlock(const unsigned char *in,
2                        unsigned char *out)
3  {
4      unsigned char current[4][4] = {{0}};
5      unsigned char *currentp = (unsigned char *)current;
6      const unsigned char *keyp;
7      for (size_t i = 0; 4 > i; i++)
8          for (size_t j = 0; 4 > j; j++)
9              current[j][i] = in[4 * i + j];
10     keyp = this->key_block->get(0);
11     for (size_t i = 0; 16 > i; i++)
12         currentp[i] ^= keyp[i];
13     for (size_t iter = 2; this->iters > iter; iter++)
14     {
15         for (size_t i = 0; 16 > i; i++)
```

```
16              currentp[i] = SBlock::direct(currentp[i]);
17          RowShifter::direct(current);
18          ColumnMixer::direct(current);
19          keyp = this->key_block->get(iter - 1);
20          for (size_t i = 0; 16 > i; i++)
21              currentp[i] ^= keyp[i];
22      }
23      for (size_t i = 0; 16 > i; i++)
24          currentp[i] = SBlock::direct(currentp[i]);
25      RowShifter::direct(current);
26      keyp = this->key_block->get(this->iters - 1);
27      for (size_t i = 0; 16 > i; i++)
28          currentp[i] ^= keyp[i];
29      for (size_t i = 0; 4 > i; i++)
30          for (size_t j = 0; 4 > j; j++)
31              out[4 * i + j] = current[j][i];
32  }
33
34  void AES::decodeBlock(const unsigned char *in,
35                        unsigned char *out)
36  {
37      unsigned char current[4][4] = {{0}};
38      unsigned char *currentp = (unsigned char *)current;
39      const unsigned char *keyp;
40      for (size_t i = 0; 4 > i; i++)
41          for (size_t j = 0; 4 > j; j++)
42              current[j][i] = in[4 * i + j];
43      keyp = this->key_block->get(this->iters - 1);
44      for (size_t i = 0; 16 > i; i++)
45          currentp[i] ^= keyp[i];
46      RowShifter::inverse(current);
47      for (size_t i = 0; 16 > i; i++)
48          currentp[i] = SBlock::inverse(currentp[i]);
49      for (size_t iter = this->iters - 2; 0 < iter; iter--)
50      {
51          keyp = this->key_block->get(iter);
52          for (size_t i = 0; 16 > i; i++)
53              currentp[i] ^= keyp[i];
54          ColumnMixer::inverse(current);
55          RowShifter::inverse(current);
56          for (size_t i = 0; 16 > i; i++)
```

```
57              currentp[i] = SBlock::inverse(currentp[i]);
58      }
59      keyp = this->key_block->get(0);
60      for (size_t i = 0; 16 > i; i++)
61          currentp[i] ^= keyp[i];
62      for (size_t i = 0; 4 > i; i++)
63          for (size_t j = 0; 4 > j; j++)
64              out[4 * i + j] = current[j][i];
65  }
```

Листинг 3.2 – Класс, реализующий получение раундовых ключей

```
 1  KeyBlock::KeyBlock(const std::string &key)
 2  {
 3      this->checkKey(key);
 4      this->keys = \
 5      std::make_unique<unsigned char[]>(this->niters * 16);
 6      memmove(this->keys.get(), key.data(), this->length);
 7      unsigned char rc = 1;
 8
 9      for (size_t block = this->length / 4,
10                  blocks = block,
11                  limit = this->niters * 4;
12           limit > block; block++)
13      {
14          unsigned char *current  = &this->keys[4 * block],
15                        *previous = current - this->length;
16          this->blockCopy(current, current - 4);
17
18          if (0 == block % blocks)
19              this->blockRc(
20                  this->blockXOR(
21                      this->blockS(this->blockShift(current)),
22                      previous),
23                  &rc);
24          else if (32 == length && 4 == block % blocks)
25              this->blockXOR(this->blockS(current), previous);
26          else
27              this->blockXOR(current, previous);
28      }
29  }
```

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)	abcdefghijklmnop <блок 16 байт>	abcdefghijklmnop \xac\x68\x64\xbe \x8e\xae\x82\x55 \x8c\x7a\x45\xcc \xd4\x8c\x6a\x42

Заключение

Была разработана программы шифрования симметричным алгоритмом AES с применением режима шифрования CFB.

Были решены следующие задачи:

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