Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Криптографические методы обеспечения информационной безопасности»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Основные структурные элементы алгоритма AES»

Выполнил:

Чу Ван Доан, студент группы N3347

(подпись)

Проверил:

Таранов Сергей Владимирович

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

Содержание

Введение

Цель: изучить основные принципы работы алгоритмы AES.

Задание

Для достижения поставленной цели необходимо решить следующие задачи:

- Проанализировать алгоритм AES
- Реализовать AES с возможностью отслеживать результаты выполнения раундов
- Выполнить отчёт.

Ход работы

1. AES (Advanced Encryption Standard)

- AES (сокращение от английского: Advanced Encryption Standard — расширенный стандарт шифрования) — это блочный алгоритм шифрования, утверждённый правительством США в качестве стандарта шифрования.

- Алгоритм основан на шифре Rijndael, разработанном двумя бельгийскими криптографами: Жоаном Дэменом и Винсентом Рейменом.

- AES работает с блоками данных размером 128 бит и поддерживает ключи длиной 128, 192 или 256 бит. Расширенные ключи создаются в процессе, называемом расширением ключа Rijndael.

- Большинство операций в алгоритме AES выполняются в ограниченном пространстве байтов. Каждый блок данных размером 128 бит разбивается на 16 байт, которые упорядочиваются в 4 столбца, по 4 байта в каждом, образуя матрицу 4x4, называемую состоянием.

- В зависимости от длины ключа — 128, 192 или 256 бит — алгоритм выполняется с разным числом раундов.

2. Построение таблицы S-box

2.1. Прямая S-box таблица

Прямая таблица S-box создаётся путём нахождения мультипликативного обратного значения в конечном поле $GF(2^8) = GF(2)[x] / (x^8 + x^4 + x^3 + x + 1)$ (поле, используемое в шифре Rijndael). Значение 0, не имеющее обратного, отображается на 0. Остальные обратные значения подвергаются аффинному преобразованию.

Формула для вычисления значений таблиц S-box и соответствующей обратной таблицы S-box:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рисунок 1 - Прямая S-box таблица

2.2. Обратная таблица S-box

Обратная таблица S-box — это просто обратное преобразование прямой S-box. Она рассчитывается путём применения обратного аффинного преобразования к входным значениям. Обратное аффинное преобразование выражается следующим образом:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Рисунок 2 - Обратная таблица S-box

3. Алгоритм генерации подключей

Процесс генерации ключей состоит из 4 шагов:

- **RotWord**: циклический сдвиг слова влево на 8 бит
- **SubBytes**
- **Rcon**: вычисление значения Rcon(i), где:

$$Rcon(i) = x^{(i-1)} \ mod \ (x^8 + x^4 + x^3 + x + 1)$$

- **ShiftRow**

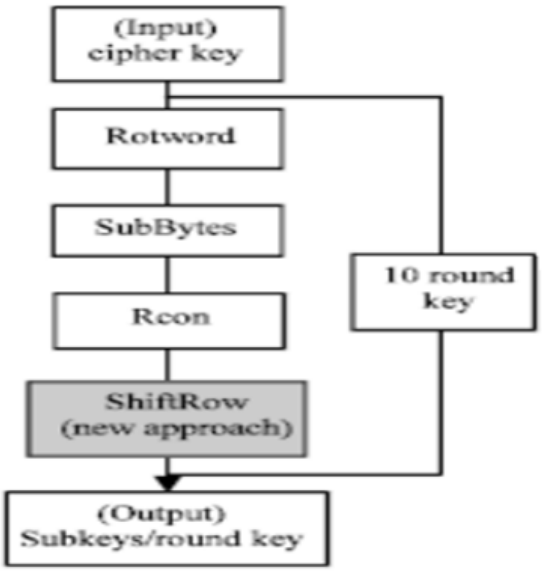

Рисунок 3 - Алгоритм генерации подключей

4. Процесс шифрования

4.1. Общая схема

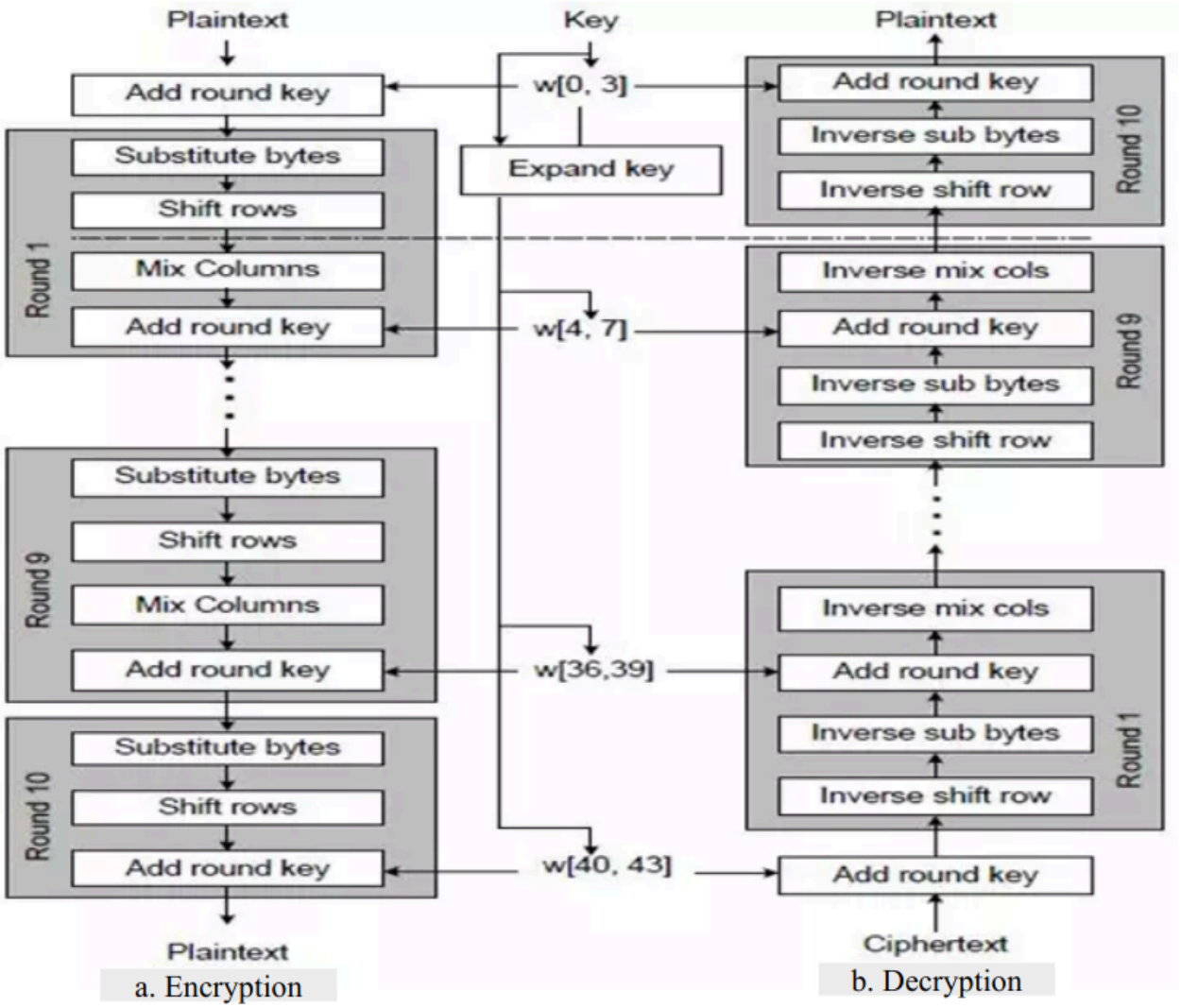

Рисунок 4 - Схема

4.2. Функция Add Round Key

- Применяется с первого раунда до раунда Nr.
- В преобразовании AddRoundKey(), раундовый ключ объединяется с состоянием с помощью побитовой операции XOR.
- Каждый раундовый ключ состоит из 4 слов (128 бит), полученных из расписания ключей. Эти 4 слова прибавляются к каждому столбцу состояния следующим образом:

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c} S_{2,c} S_{3,c}] \oplus W(4i + c), \text{ где } 0 \leq c < 4$$

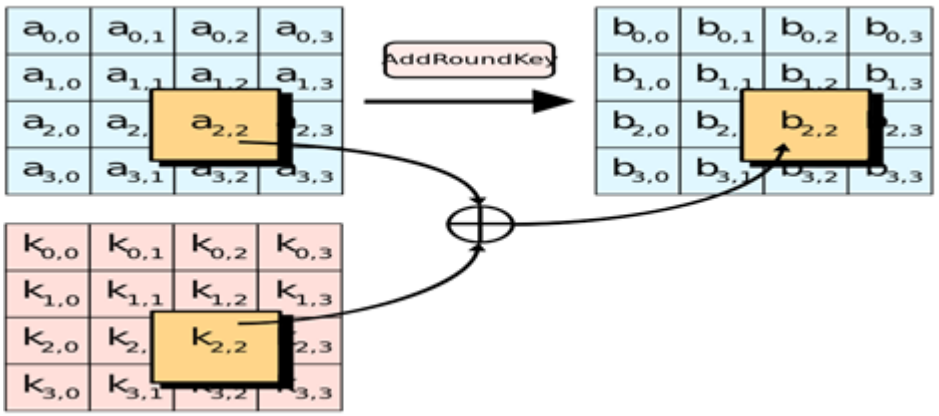

Рисунок 5 - Функция Add Round Key

4.3. Функция SubBytes

Преобразование SubBytes() заменяет каждый отдельный байт состояния $S_{r,c}$ новым значением $S'_{r,c}$, используя таблицу замены (S-box), построенную выше.

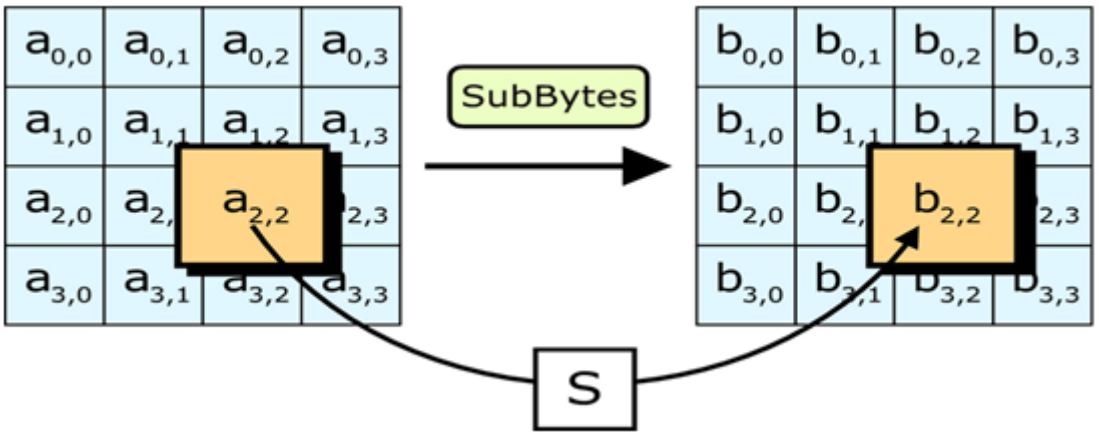

Рисунок 6 - Функция SubBytes

4.4. Функция ShiftRow

В преобразовании ShiftRows() байты в трёх последних строках состояния циклически сдвигаются на различное количество позиций (смещения). Конкретно:

$$S'_{r,c} = S_{r,(c+shift(r,Nb))} \; mod \; Nb, \quad \text{где } Nb = 4$$

Значение смещения shift(r, Nb) зависит от номера строки *r* r следующим образом:

$$shift(1,4) = 1, \quad shift(2,4) = 2, \quad shift(3,4) = 3$$

Первая строка не сдвигается, остальные сдвигаются соответственно:

- 1-я строка остаётся без изменений.
- 2-я строка сдвигается влево на 1 байт.
- 3-я строка сдвигается влево на 2 байта.
- 4-я строка сдвигается влево на 3 байта.

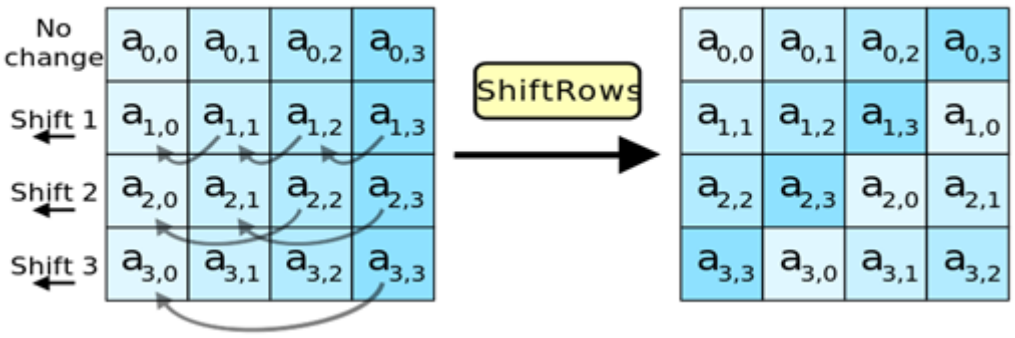

Рисунок 7 - Функция ShiftRows

4.5. Функция MixColumns

Преобразование MixColumns() применяется к каждому столбцу состояния. Каждый столбец рассматривается как многочлен в поле $GF(2^8)$и умножается на многочлен a (x), заданный следующим образом:

$$a(x) = (03)x^3 + (01)x^2 + (01)x + (02)$$

Это преобразование можно представить как умножение матрицы, где каждый байт рассматривается как элемент поля $GF(2^8)$:

$$s'(x) = a(x) * s(x)$$

Матрица описывается следующим образом:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Рисунок 7 - Функция MixColumns

5. Процесс дешифрования

Алгоритм деширования во многом схож с алгоритмом шифрования по своей структуре, но использует 4 функции, которые являются обратными к функциям шифрования.

Шифрование	Дешифрование
AddRoundKey()	InvAddRoundKey()
SubBytes()	InvSubBytes()
ShiftRows()	InvShiftRows()
MixColumns()	InvMixColumns()

6. Демонстрация работы

6.1. Encryption

```
python3 encryption.py
Round 0 (AddRoundKey):
Initial:
  32 88 31 e0
  43 5a 31 37
  f6 30 98 07
  a8 8d a2 34

Round 0 Key:
  2b 28 ab 09
  7e ae f7 cf
  15 d2 15 4f
  16 a6 88 3c

After AddRoundKey:
  19 a0 9a e9
  3d f4 c6 f8
  e3 e2 8d 48
  be 2b 2a 08

--- Round 1 ---
```

```
After SubBytes:
  d4 e0 b8 1e
  27 bf b4 41
  11 98 5d 52
  ae f1 e5 30

After ShiftRows:
  d4 e0 b8 1e
  bf b4 41 27
  5d 52 11 98
  30 ae f1 e5

After MixColumns:
  04 e0 48 28
  66 cb f8 06
  81 19 d3 26
  e5 9a 7a 4c

Round 1 Key:
  a0 88 23 2a
  fa 54 a3 6c
  fe 2c 39 76
  17 b1 39 05

After AddRoundKey (Round 1):
  a4 68 6b 02
  9c 9f 5b 6a
  7f 35 ea 50
  f2 2b 43 49

--- Round 2 ---
After SubBytes:
  49 45 7f 77
  de db 39 02
  d2 96 87 53
  89 f1 1a 3b

After ShiftRows:
  49 45 7f 77
  db 39 02 de
  87 53 d2 96
  3b 89 f1 1a
```

```
After MixColumns:
  58 1b db 1b
  4d 4b e7 6b
  ca 5a ca b0
  f1 ac a8 e5

Round 2 Key:
  f2 7a 59 73
  c2 96 35 59
  95 b9 80 f6
  f2 43 7a 7f

After AddRoundKey (Round 2):
  aa 61 82 68
  8f dd d2 32
  5f e3 4a 46
  03 ef d2 9a

--- Round 3 ---
After SubBytes:
  ac ef 13 45
  73 c1 b5 23
  cf 11 d6 5a
  7b df b5 b8

After ShiftRows:
  ac ef 13 45
  c1 b5 23 73
  d6 5a cf 11
  b8 7b df b5

After MixColumns:
  75 20 53 bb
  ec 0b c0 25
  09 63 cf d0
  93 33 7c dc

Round 3 Key:
  3d 47 1e 6d
  80 16 23 7a
  47 fe 7e 88
  7d 3e 44 3b
```

```
After AddRoundKey (Round 3):
  48 67 4d d6
  6c 1d e3 5f
  4e 9d b1 58
  ee 0d 38 e7

--- Round 4 ---
After SubBytes:
  52 85 e3 f6
  50 a4 11 cf
  2f 5e c8 6a
  28 d7 07 94

After ShiftRows:
  52 85 e3 f6
  a4 11 cf 50
  c8 6a 2f 5e
  94 28 d7 07

After MixColumns:
  0f 60 6f 5e
  d6 31 c0 b3
  da 38 10 13
  a9 bf 6b 01

Round 4 Key:
  ef a8 b6 db
  44 52 71 0b
  a5 5b 25 ad
  41 7f 3b 00

After AddRoundKey (Round 4):
  e0 c8 d9 85
  92 63 b1 b8
  7f 63 35 be
  e8 c0 50 01

--- Round 5 ---
After SubBytes:
  e1 e8 35 97
  4f fb c8 6c
  d2 fb 96 ae
  9b ba 53 7c
```

```
After ShiftRows:
  e1 e8 35 97
  fb c8 6c 4f
  96 ae d2 fb
  7c 9b ba 53

After MixColumns:
  25 bd b6 4c
  d1 11 3a 4c
  a9 d1 33 c0
  ad 68 8e b0

Round 5 Key:
  d4 7c ca 11
  d1 83 f2 f9
  c6 9d b8 15
  f8 87 bc bc

After AddRoundKey (Round 5):
  f1 c1 7c 5d
  00 92 c8 b5
  6f 4c 8b d5
  55 ef 32 0c

--- Round 6 ---
After SubBytes:
  a1 78 10 4c
  63 4f e8 d5
  a8 29 3d 03
  fc df 23 fe

After ShiftRows:
  a1 78 10 4c
  4f e8 d5 63
  3d 03 a8 29
  fe fc df 23

After MixColumns:
  4b 2c 33 37
  86 4a 9d d2
  8d 89 f4 18
  6d 80 e8 d8
```

```
Round 6 Key:
  6d 11 db ca
  88 0b f9 00
  a3 3e 86 93
  7a fd 41 fd

After AddRoundKey (Round 6):
  26 3d e8 fd
  0e 41 64 d2
  2e b7 72 8b
  17 7d a9 25

--- Round 7 ---
After SubBytes:
  f7 27 9b 54
  ab 83 43 b5
  31 a9 40 3d
  f0 ff d3 3f

After ShiftRows:
  f7 27 9b 54
  83 43 b5 ab
  40 3d 31 a9
  3f f0 ff d3

After MixColumns:
  14 46 27 34
  15 16 46 2a
  b5 15 56 d8
  bf ec d7 43

Round 7 Key:
  4e 5f 84 4e
  54 5f a6 a6
  f7 c9 4f dc
  0e f3 b2 4f

After AddRoundKey (Round 7):
  5a 19 a3 7a
  41 49 e0 8c
  42 dc 19 04
  b1 1f 65 0c
```

```
--- Round 8 ---
After SubBytes:
  be d4 0a da
  83 3b e1 64
  2c 86 d4 f2
  c8 c0 4d fe

After ShiftRows:
  be d4 0a da
  3b e1 64 83
  d4 f2 2c 86
  fe c8 c0 4d

After MixColumns:
  00 b1 54 fa
  51 c8 76 1b
  2f 89 6d 99
  d1 ff cd ea

Round 8 Key:
  ea b5 31 7f
  d2 8d 2b 8d
  73 ba f5 29
  21 d2 60 2f

After AddRoundKey (Round 8):
  ea 04 65 85
  83 45 5d 96
  5c 33 98 b0
  f0 2d ad c5

--- Round 9 ---
After SubBytes:
  87 f2 4d 97
  ec 6e 4c 90
  4a c3 46 e7
  8c d8 95 a6

After ShiftRows:
  87 f2 4d 97
  6e 4c 90 ec
  46 e7 4a c3
```

```
  a6 8c d8 95

After MixColumns:
  47 40 a3 4c
  37 d4 70 9f
  94 e4 3a 42
  ed a5 a6 bc

Round 9 Key:
  ac 19 28 57
  77 fa d1 5c
  66 dc 29 00
  f3 21 41 6e

After AddRoundKey (Round 9):
  eb 59 8b 1b
  40 2e a1 c3
  f2 38 13 42
  1e 84 e7 d2

--- Final Round ---
After SubBytes:
  e9 cb 3d af
  09 31 32 2e
  89 07 7d 2c
  72 5f 94 b5

After ShiftRows:
  e9 cb 3d af
  31 32 2e 09
  7d 2c 89 07
  b5 72 5f 94

Round 10 Key:
  d0 c9 e1 b6
  14 ee 3f 63
  f9 25 0c 0c
  a8 89 c8 a6

After Final AddRoundKey:
  39 02 dc 19
  25 dc 11 6a
  84 09 85 0b
```

```
  1d fb 97 32


Ciphertext:
['0x39', '0x25', '0x84', '0x1d', '0x2', '0xdc', '0x9', '0xfb', '0xdc',
'0x11', '0x85', '0x97', '0x19', '0x6a', '0xb', '0x32']
```

6.2. Decryption

```
--- Final Round ---
Round 0 Key:
   2b 28 ab 09
   7e ae f7 cf
   15 d2 15 4f
   16 a6 88 3c

After Final AddRoundKey:
   32 88 31 e0
   43 5a 31 37
   f6 30 98 07
   a8 8d a2 34


Decrypted Plaintext: ['0x32', '0x43', '0xf6', '0xa8', '0x88', '0x5a', '0x30', '0x8d', '0x31', '0x31', '0x98', '0xa2', '0xe0', '0x37', '0x7', '0x34']
```

Рисунок 8 - Decryption

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- Проанализирован алгоритм AES
- Реализован алгоритм AES
- Выполнен отчёт

Таким образом, все поставленные задачи решены, цель работы успешно достигнута.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Код файла encryption.py

```
# AES-128 Step-by-step Encryption
import copy

# AES S-box
s_box = [
    # 0     1     2     3     4     5     6     7     8     9     A
B     C     D     E     F
    [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76], # 0
    [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0], # 1
    [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15], # 2
    [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75], # 3
    [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84], # 4
    [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf], # 5
    [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8], # 6
    [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2], # 7
    [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73], # 8
    [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb], # 9
    [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79], # A
    [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08], # B
    [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a], # C
    [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e], # D
    [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf], # E
    [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16]  # F
```

```
]

# Rcon cho Key Expansion
r_con = [
    0x00, 0x01, 0x02, 0x04, 0x08,
    0x10, 0x20, 0x40, 0x80, 0x1B, 0x36
]

def sub_bytes(state):
    for i in range(4):
        for j in range(4):
            byte = state[i][j]
            row, col = byte >> 4, byte & 0x0F
            state[i][j] = s_box[row][col]
    return state

def shift_rows(state):
    for i in range(1, 4):
        state[i] = state[i][i:] + state[i][:i]
    return state

def xtime(a):
    return ((a << 1) ^ 0x1B) & 0xFF if (a & 0x80) else (a << 1)

def mix_single_column(col):
    t = col[0] ^ col[1] ^ col[2] ^ col[3]
    u = col[0]
    col[0] ^= t ^ xtime(col[0] ^ col[1])
    col[1] ^= t ^ xtime(col[1] ^ col[2])
    col[2] ^= t ^ xtime(col[2] ^ col[3])
    col[3] ^= t ^ xtime(col[3] ^ u)
    return col

def mix_columns(state):
    for i in range(4):
        col = [state[j][i] for j in range(4)]
        col = mix_single_column(col)
        for j in range(4):
            state[j][i] = col[j]
    return state

def add_round_key(state, round_key):
    for i in range(4):
```

```
        for j in range(4):
            state[i][j] ^= round_key[i][j]
    return state

def key_expansion(key):
    key_symbols = [b for b in key]
    key_schedule = [[] for _ in range(4)]
    for r in range(4):
        for c in range(4):
            key_schedule[r].append(key_symbols[r + 4 * c])
    for i in range(4, 44):
        temp = [key_schedule[j][i - 1] for j in range(4)]
        if i % 4 == 0:
            temp = temp[1:] + temp[:1]  # RotWord
            for j in range(4):
                byte = temp[j]
                row, col = byte >> 4, byte & 0x0F
                temp[j] = s_box[row][col]  # SubWord
            temp[0] ^= r_con[i // 4]
        for j in range(4):
            temp[j] ^= key_schedule[j][i - 4]
            key_schedule[j].append(temp[j])
    round_keys = []
    for i in range(0, 44, 4):
        round_keys.append([[key_schedule[r][i + c] for c in range(4)] for r
in range(4)])
    return round_keys

def print_state(state, label="State"):
    print(f"{label}:")
    for row in state:
        print("  ", ' '.join(f"{b:02x}" for b in row))
    print()

def print_round_key(round_key, round_num):
    print(f"Round {round_num} Key:")
    for row in round_key:
        print("  ", ' '.join(f"{b:02x}" for b in row))
    print()

def aes_encrypt(plaintext, key):
    state = [[plaintext[r + 4 * c] for c in range(4)] for r in range(4)]
    round_keys = key_expansion(key)
```

```
    print("Round 0 (AddRoundKey):")
    print_state(state, "Initial")
    print_round_key(round_keys[0], 0)
    state = add_round_key(state, round_keys[0])
    print_state(state, "After AddRoundKey")

    for rnd in range(1, 10):
        print(f"--- Round {rnd} ---")
        state = sub_bytes(state)
        print_state(state, "After SubBytes")

        state = shift_rows(state)
        print_state(state, "After ShiftRows")

        state = mix_columns(state)
        print_state(state, "After MixColumns")

        print_round_key(round_keys[rnd], rnd)
        state = add_round_key(state, round_keys[rnd])
        print_state(state, f"After AddRoundKey (Round {rnd})")

    print("--- Final Round ---")
    state = sub_bytes(state)
    print_state(state, "After SubBytes")

    state = shift_rows(state)
    print_state(state, "After ShiftRows")

    print_round_key(round_keys[10], 10)
    state = add_round_key(state, round_keys[10])
    print_state(state, "After Final AddRoundKey")

    ciphertext = [state[r][c] for c in range(4) for r in range(4)]
    return ciphertext

# Example usage
plaintext = [0x32, 0x43, 0xf6, 0xa8,
             0x88, 0x5a, 0x30, 0x8d,
             0x31, 0x31, 0x98, 0xa2,
             0xe0, 0x37, 0x07, 0x34]

key = [0x2b, 0x7e, 0x15, 0x16,
```

```
        0x28, 0xae, 0xd2, 0xa6,
        0xab, 0xf7, 0x15, 0x88,
        0x09, 0xcf, 0x4f, 0x3c]

ciphertext = aes_encrypt(plaintext, key)
print("\nCiphertext:")
print([hex(b) for b in ciphertext])
```

Листинг А.2 – Код файла decryption.py

```
# AES-128 Decryption (detailed output)

import copy

# AES S-box
s_box = [
    # 0      1      2      3      4      5      6      7      8      9      A
B      C      D      E      F
    [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76], # 0
    [0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0], # 1
    [0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15], # 2
    [0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75], # 3
    [0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84], # 4
    [0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf], # 5
    [0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8], # 6
    [0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2], # 7
    [0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73], # 8
    [0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb], # 9
    [0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79], # A
    [0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08], # B
```

```
    [0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a], # C
    [0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e], # D
    [0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf], # E
    [0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16]  # F
]

r_con = [
    0x00, 0x01, 0x02, 0x04, 0x08,
    0x10, 0x20, 0x40, 0x80, 0x1B, 0x36
]


inv_s_box = [[0 for _ in range(16)] for _ in range(16)]
for i in range(256):
    row, col = i >> 4, i & 0x0F
    val = s_box[row][col]
    inv_s_box[val >> 4][val & 0x0F] = i

def xtime(a):
    return ((a << 1) ^ 0x1B) & 0xFF if (a & 0x80) else (a << 1)

def mix_single_column(col):
    t = col[0] ^ col[1] ^ col[2] ^ col[3]
    u = col[0]
    col[0] ^= t ^ xtime(col[0] ^ col[1])
    col[1] ^= t ^ xtime(col[1] ^ col[2])
    col[2] ^= t ^ xtime(col[2] ^ col[3])
    col[3] ^= t ^ xtime(col[3] ^ u)
    return col

def print_state(state, label="State"):
    print(f"{label}:")
    for row in state:
        print("  ", ' '.join(f"{b:02x}" for b in row))
    print()

def print_round_key(round_key, round_num):
    print(f"Round {round_num} Key:")
    for row in round_key:
```

```
        print("  ", ' '.join(f"{b:02x}" for b in row))
    print()

def add_round_key(state, round_key):
    for i in range(4):
        for j in range(4):
            state[i][j] ^= round_key[i][j]
    return state

# --- Giải mã các bước ---
def inv_sub_bytes(state):
    for i in range(4):
        for j in range(4):
            byte = state[i][j]
            row, col = byte >> 4, byte & 0x0F
            state[i][j] = inv_s_box[row][col]
    return state

def inv_shift_rows(state):
    for i in range(1, 4):
        state[i] = state[i][-i:] + state[i][:-i]
    return state

def inv_mix_columns(state):
    for i in range(4):
        col = [state[j][i] for j in range(4)]
        u = xtime(xtime(col[0] ^ col[2]))
        v = xtime(xtime(col[1] ^ col[3]))
        col[0] ^= u
        col[1] ^= v
        col[2] ^= u
        col[3] ^= v
        col = mix_single_column(col)
        for j in range(4):
            state[j][i] = col[j]
    return state

# --- Key Expansion ---
def key_expansion(key):
    key_symbols = [b for b in key]
    key_schedule = [[] for _ in range(4)]
    for r in range(4):
        for c in range(4):
```

```
            key_schedule[r].append(key_symbols[r + 4 * c])
    for i in range(4, 44):
        temp = [key_schedule[j][i - 1] for j in range(4)]
        if i % 4 == 0:
            temp = temp[1:] + temp[:1]
            for j in range(4):
                byte = temp[j]
                row, col = byte >> 4, byte & 0x0F
                temp[j] = s_box[row][col]
            temp[0] ^= r_con[i // 4]
        for j in range(4):
            temp[j] ^= key_schedule[j][i - 4]
            key_schedule[j].append(temp[j])
    round_keys = []
    for i in range(0, 44, 4):
        round_keys.append([[key_schedule[r][i + c] for c in range(4)] for r
in range(4)])
    return round_keys

# --- AES Decrypt ---
def aes_decrypt(ciphertext, key):
    state = [[ciphertext[r + 4 * c] for c in range(4)] for r in range(4)]
    round_keys = key_expansion(key)

    print("Initial Ciphertext State:")
    print_state(state)

    print_round_key(round_keys[10], 10)
    state = add_round_key(state, round_keys[10])
    print_state(state, "After Initial AddRoundKey")

    state = inv_shift_rows(state)
    print_state(state, "After InvShiftRows")

    state = inv_sub_bytes(state)
    print_state(state, "After InvSubBytes")

    for rnd in range(9, 0, -1):
        print(f"--- Round {rnd} ---")
        print_round_key(round_keys[rnd], rnd)
        state = add_round_key(state, round_keys[rnd])
        print_state(state, "After AddRoundKey")
```

```
            state = inv_mix_columns(state)
            print_state(state, "After InvMixColumns")

            state = inv_shift_rows(state)
            print_state(state, "After InvShiftRows")

            state = inv_sub_bytes(state)
            print_state(state, "After InvSubBytes")

        print("--- Final Round ---")
        print_round_key(round_keys[0], 0)
        state = add_round_key(state, round_keys[0])
        print_state(state, "After Final AddRoundKey")

        plaintext = [state[r][c] for c in range(4) for r in range(4)]
        return plaintext

    if __name__ == "__main__":
        plaintext = [0x32, 0x43, 0xf6, 0xa8,
                     0x88, 0x5a, 0x30, 0x8d,
                     0x31, 0x31, 0x98, 0xa2,
                     0xe0, 0x37, 0x07, 0x34]

        key = [0x2b, 0x7e, 0x15, 0x16,
               0x28, 0xae, 0xd2, 0xa6,
               0xab, 0xf7, 0x15, 0x88,
               0x09, 0xcf, 0x4f, 0x3c]

        from encryption import aes_encrypt

        ciphertext = aes_encrypt(plaintext, key)
        print("\nCiphertext:", [hex(b) for b in ciphertext])

        decrypted = aes_decrypt(ciphertext, key)
        print("\nDecrypted Plaintext:", [hex(b) for b in decrypted])
```