**МИНОБРНАУКИ РОССИИ ФЕДЕРАЛЬНОЕ**

**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО**

**ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ**

**УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГУ»)**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ, ИНФОРМАТИКИ И**

**МЕХАНИКИ**

Дисциплина «Безопасность мобильных устройств»

Направление 02.04.02 «Фундаментальная информатика и

информационные технологии»

Лабораторная работа №2

Выполнила магистры 2 курса 13 группы:

Бутузова Д. О.

Преподаватель: Вернер Е. С.

Воронеж 2025

### Введение

В данном отчёте рассматривается алгоритм симметричного блочного шифрования AES и его реализация на языке программирования Python.

# Цели

1. Познакомиться с алгоритмом симметричного блочного шифрования AES;
2. Реализовать его на языке программирования Python.

# Алгоритм AES

AES (Advanced Encryption Standard) — это симметричный блочный алгоритм шифрования, утверждённый как стандарт шифрования правительством США в 2001 году. Он заменил устаревший DES.

**Основные характеристики AES-128:**

* Блочная длина: 128 бит (16 байт);
* Ключ: 128 бит;
* Количество раундов: 10.

Каждый раунд (кроме последнего) включает:

* **SubBytes** — байтовая подстановка с использованием таблицы S-box;
* **ShiftRows** — циклический сдвиг строк матрицы состояния;
* **MixColumns** — линейная трансформация столбцов;
* **AddRoundKey** — побитовая операция XOR с раундовым ключом.

Последний раунд исключает MixColumns.

# Разработка программы на языке Python

Была реализована функция шифрования и расшифровки 16-байтного блока данных по стандарту AES-128.

Основные этапы:

* Реализация S-box и обратной S-box;
* Функции SubBytes, ShiftRows, MixColumns и их обратные;
* Генерация раундовых ключей (key expansion);
* Реализация шифрования (encrypt\_block) и расшифровки (decrypt\_block).

## Реализация S-box

Алгоритм генерации S-box:

1. Возьмём все числа от 0 до 255.
2. Для каждого числа находим его обратное значение в поле GF(2^8)(поле Галуа GF(2^8) (множество 256 элементов)).
3. Применяем афинное преобразование для улучшения стойкости против атак:
   * + 1. Умножение на фиксированное значение (обычно используется полином x^8 + x^4 + x^3 + x + 1).
       2. **XOR** с фиксированной константой для усиления случайности.
4. Помещаем полученные значения в таблицу.

Основные функции:

* galois\_mult(x, y) — функция для умножения в поле Галуа GF(2^8).
* galois\_inv(x) — находит обратный элемент в поле Галуа.
* affine\_transformation(x) — выполняет афинное преобразование для каждого байта, используя XOR с фиксированным значением.
* generate\_s\_box() — генерирует таблицу S-box, применяя обратный элемент и афинное преобразование для каждого байта от 0 до 255.

Реализация данных функций на языке python добавлена в приложении к отчету.

S-box:

00 0e 1d 13 3b 35 26 28 77 79 6a 64 4c 42 51 5f

ee e0 f3 fd d5 db c8 c6 99 97 84 8a a2 ac bf b1

d6 d8 cb c5 ed e3 f0 fe a1 af bc b2 9a 94 87 89

38 36 25 2b 03 0d 1e 10 4f 41 52 5c 74 7a 69 67

a7 a9 ba b4 9c 92 81 8f d0 de cd c3 eb e5 f6 f8

49 47 54 5a 72 7c 6f 61 3e 30 23 2d 05 0b 18 16

71 7f 6c 62 4a 44 57 59 06 08 1b 15 3d 33 20 2e

9f 91 82 8c a4 aa b9 b7 e8 e6 f5 fb d3 dd ce c0

45 4b 58 56 7e 70 63 6d 32 3c 2f 21 09 07 14 1a

ab a5 b6 b8 90 9e 8d 83 dc d2 c1 cf e7 e9 fa f4

93 9d 8e 80 a8 a6 b5 bb e4 ea f9 f7 df d1 c2 cc

7d 73 60 6e 46 48 5b 55 0a 04 17 19 31 3f 2c 22

e2 ec ff f1 d9 d7 c4 ca 95 9b 88 86 ae a0 b3 bd

0c 02 11 1f 37 39 2a 24 7b 75 66 68 40 4e 5d 53

34 3a 29 27 0f 01 12 1c 43 4d 5e 50 78 76 65 6b

da d4 c7 c9 e1 ef fc f2 ad a3 b0 be 96 98 8b 85

## Реализация обратного S-box

Обратный S-box используется в процессе расшифровки данных. Процесс его создания аналогичен прямому S-box, но с некоторыми отличиями.

Алгоритм генерации обратного S-box:

1. **Вместо обратного элемента для каждого числа мы используем прямой S-box.** В обратном S-box позиции значений из прямого S-box меняются местами, то есть каждый элемент обратного S-box указывает на индекс, который в прямом S-box соответствует этому элементу.
2. **Инвертируем индексы и значения.** Мы создаём новый массив (обратный S-box), где каждый элемент представляет собой индекс значения в прямом S-box.
3. **Преобразованные значения из прямого S-box становятся обратными.** Например, если элемент в прямом S-box имеет значение 0x53 на позиции 0x42, то в обратном S-box на позиции 0x53 будет стоять значение 0x42.
4. Помещаем полученные значения в таблицу.

Таким образом, обратный S-box — это просто инвертированная версия прямого S-box, где индексы и значения меняются местами.

Функция generate\_inv\_s\_box(s\_box) используется для обратного S-box. Применяется простая инверсия значений в таблице S-box: каждый элемент прямого S-box устанавливается на соответствующую позицию в обратной таблице, так что при дешифровке мы можем использовать его для восстановления исходного текста.

Inverse S-box:

00 e5 d1 34 b9 5c 68 8d 69 8c b8 5d d0 35 01 e4

37 d2 e6 03 8e 6b 5f ba 5e bb 8f 6a e7 02 36 d3

6e 8b bf 5a d7 32 06 e3 07 e2 d6 33 be 5b 6f 8a

59 bc 88 6d e0 05 31 d4 30 d5 e1 04 89 6c 58 bd

dc 39 0d e8 65 80 b4 51 b5 50 64 81 0c e9 dd 38

eb 0e 3a df 52 b7 83 66 82 67 53 b6 3b de ea 0f

b2 57 63 86 0b ee da 3f db 3e 0a ef 62 87 b3 56

85 60 54 b1 3c d9 ed 08 ec 09 3d d8 55 b0 84 61

a3 46 72 97 1a ff cb 2e ca 2f 1b fe 73 96 a2 47

94 71 45 a0 2d c8 fc 19 fd 18 2c c9 44 a1 95 70

cd 28 1c f9 74 91 a5 40 a4 41 75 90 1d f8 cc 29

fa 1f 2b ce 43 a6 92 77 93 76 42 a7 2a cf fb 1e

7f 9a ae 4b c6 23 17 f2 16 f3 c7 22 af 4a 7e 9b

48 ad 99 7c f1 14 20 c5 21 c4 f0 15 98 7d 49 ac

11 f4 c0 25 a8 4d 79 9c 78 9d a9 4c c1 24 10 f5

26 c3 f7 12 9f 7a 4e ab 4f aa 9e 7b f6 13 27 c2

## Реализация алгоритма AES

Основные этапы алгоритма AES:

1. Инициализация:
   * + Преобразование исходного текста в состояние (state) размером 4x4, которое представляет собой 128 бит (16 байт).
     + Сгенерированные раундовые ключи, полученные из основного ключа, будут использоваться на каждом этапе шифрования.
2. Раунды шифрования: Алгоритм AES использует 10 раундов (для 128-битного ключа), каждый из которых состоит из следующих шагов:
   * + AddRoundKey: К каждому байту состояния применяется операция XOR с соответствующим раундовым ключом.
     + SubBytes: Каждый байт состояния заменяется согласно таблице S-box (для прямого шифрования).
     + ShiftRows: Каждый ряд в состоянии сдвигается на определённое количество позиций.
     + MixColumns: Каждый столбец состояния преобразуется с помощью матричного умножения в поле Галуа (для всех раундов, кроме последнего).
     + AddRoundKey: Добавление раундового ключа (после всех шагов кроме последнего).
3. Последний раунд: В последнем раунде исключается шаг MixColumns.
4. Дешифрование: Процесс дешифрования практически идентичен шифрованию, но используется обратный порядок шагов. В частности:
   * + Вместо таблицы S-box используется обратный S-box (для обратного шифрования).
     + Вместо MixColumns используется операция InvMixColumns.
     + Порядок операций в раунде изменён (обратный).

Основные функции шифрования:

* key\_expansion(key): Расширение ключа. Процесс, в котором из основного ключа генерируются все раундовые ключи, используемые в алгоритме. Для 128-битного ключа генерируется 11 раундовых ключей.
* add\_round\_key(state, round\_key): Добавление раундового ключа. Каждый байт состояния XOR'ится с соответствующим байтом раундового ключа.
* sub\_bytes(state, s\_box): Подстановка байтов. Каждый байт состояния заменяется согласно таблице S-box.
* shift\_rows(state): Сдвиг строк. Каждая строка состояния сдвигается на определённое количество позиций.
* mix\_columns(state): Смешивание столбцов. Каждый столбец состояния преобразуется с использованием операции в поле Галуа (для всех раундов, кроме последнего).

Основные функции шифрования:

* def inv\_sub\_bytes(state, inv\_s\_box): Обратная подстановка байтов. Используется обратный S-box для замены байтов в процессе расшифровки.
* def inv\_shift\_rows(state): Обратный сдвиг строк. Строки состояния сдвигаются в обратном порядке.
* def inv\_mix\_columns(state): Обратное смешивание столбцов. Обратное смешивание столбцов в процессе расшифровки.
* def add\_round\_key(state, round\_key): Добавление раундового ключа. Точно так же, как и в шифровании, в процессе дешифровки добавляется раундовый ключ с помощью операции XOR.

# Пример работы программы

Исходное сообщение: b'Top secret info!'

Ключ: b'And this is key'

Зашифрованный блок:

e6003606c143363d43e5e1ea1b3abcb4

Расшифрованный блок:

b'Top secret info!'

### Вывод

В ходе выполнения лабораторной работы была исследована реализация алгоритма симметричного шифрования AES, который является одним из самых популярных стандартов шифрования данных. В процессе работы были выполнены следующие ключевые задачи:

* **Реализация алгоритма AES**
* **Генерация таблиц S-box и Inverse S-box**
* **Реализация шифрования и дешифрования**
* **Проверка работы алгоритма**

Лабораторная работа позволила ознакомиться с принципами работы алгоритма AES, реализовать его с использованием основных операций шифрования и дешифрования. Реализованный алгоритм позволяет обеспечивать надежное шифрование данных с использованием симметричного ключа, что делает его эффективным инструментом для защиты конфиденциальной информации в различных приложениях.

### Приложение

Ссылка на репозиторий на GitHub <https://github.com/darya002/bmu_lab2>

Файл constants.py

m = 0x11b  
def galois\_mult(x, y):  
 result = 0  
 while y > 0:  
 if y & 1:  
 result ^= x  
 y >>= 1  
 x <<= 1  
 if x >= 0x100:  
 x ^= m  
 return result  
  
def galois\_inv(x):  
 return galois\_mult(x, 0x0e)  
  
def affine\_transformation(x):  
 return ((x >> 4) ^ (x)) & 0xff  
  
def generate\_s\_box():  
 s\_box = []  
 for x in range(256):  
 if x == 0:  
 s\_box.append(0)  
 else:  
 inv = galois\_inv(x)  
 s\_box.append(affine\_transformation(inv))  
 return s\_box  
  
def print\_s\_box(s\_box):  
 print("S-box:")  
 for i in range(0, 256, 16):  
 row = " ".join([f"{s\_box[j]:02x}" for j in range(i, i+16)])  
 print(row)  
  
s\_box = generate\_s\_box()  
print\_s\_box(s\_box)  
  
def generate\_inv\_s\_box(s\_box):  
 inv\_s\_box = [0] \* 256  
 for i, value in enumerate(s\_box):  
 inv\_s\_box[value] = i  
 return inv\_s\_box  
  
def print\_inv\_s\_box(inv\_s\_box):  
 print("Inverse S-box:")  
 for i in range(0, 256, 16):  
 row = " ".join([f"{inv\_s\_box[j]:02x}" for j in range(i, i+16)])  
 print(row)  
  
r\_con = [  
 0x01, 0x02, 0x04, 0x08,  
 0x10, 0x20, 0x40, 0x80,  
 0x1B, 0x36  
]

Генерация ключей, файл key\_expansion.py

def sub\_word(word):  
 return [s\_box[b] for b in word]  
  
def rot\_word(word):  
 return word[1:] + word[:1]  
  
def key\_expansion(key):  
 key\_symbols = [b for b in key]  
 if len(key\_symbols) != 16:  
 raise ValueError("Key must be 16 bytes long for AES-128")  
  
 # Первые 4 слова = исходный ключ  
 w = [key\_symbols[i:i + 4] for i in range(0, 16, 4)]  
  
 for i in range(4, 44):  
 temp = w[i - 1]  
 if i % 4 == 0:  
 temp = sub\_word(rot\_word(temp))  
 temp[0] ^= r\_con[i // 4 - 1]  
 word = [w[i - 4][j] ^ temp[j] for j in range(4)]  
 w.append(word)  
  
 return w  
def print\_round\_keys(expanded\_key):  
 print("Раундовые ключи:")  
 for i in range(11): # 11 ключей для 10 раундов + начальный  
 key = expanded\_key[i\*4:(i+1)\*4]  
 flat = sum(key, [])  
 print(f"Round {i:2}: {' '.join(f'{b:02x}' for b in flat)}")

Шифрование, файл aes\_encription.py

def sub\_bytes(state):  
 for i in range(4):  
 for j in range(4):  
 state[i][j] = s\_box[state[i][j]]  
 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)]  
 mixed = mix\_single\_column(col)  
 for j in range(4):  
 state[j][i] = mixed[j]  
 return state  
  
def add\_round\_key(state, round\_key):  
 for i in range(4): # строки  
 for j in range(4): # столбцы  
 state[i][j] ^= round\_key[j][i]  
 return state  
  
def encrypt\_block(block, expanded\_key):  
 state = [[0]\*4 for \_ in range(4)]  
  
 for i in range(16):  
 state[i % 4][i // 4] = block[i]  
  
 round\_keys = [expanded\_key[i\*4:(i+1)\*4] for i in range(11)]  
 state = add\_round\_key(state, round\_keys[0])  
 for rnd in range(1, 10):  
 state = sub\_bytes(state)  
 state = shift\_rows(state)  
 state = mix\_columns(state)  
 state = add\_round\_key(state, round\_keys[rnd])  
  
 state = sub\_bytes(state)  
 state = shift\_rows(state)  
 state = add\_round\_key(state, round\_keys[10])  
  
 output = []  
 for i in range(4):  
 for j in range(4):  
 output.append(state[j][i])  
 return bytes(output)

Шифрование, файл aes\_decription.py

def inv\_sub\_bytes(state):  
 for i in range(4):  
 for j in range(4):  
 state[i][j] = inv\_s\_box[state[i][j]]  
 return state  
  
def inv\_shift\_rows(state):  
 for i in range(1, 4):  
 state[i] = state[i][-i:] + state[i][:-i]  
 return state  
  
def mul(a, b):  
 p = 0  
 for i in range(8):  
 if b & 1:  
 p ^= a  
 hi\_bit\_set = a & 0x80  
 a = (a << 1) & 0xFF  
 if hi\_bit\_set:  
 a ^= 0x1B  
 b >>= 1  
 return p  
  
def inv\_mix\_single\_column(col):  
 c0 = mul(col[0], 0x0e) ^ mul(col[1], 0x0b) ^ mul(col[2], 0x0d) ^ mul(col[3], 0x09)  
 c1 = mul(col[0], 0x09) ^ mul(col[1], 0x0e) ^ mul(col[2], 0x0b) ^ mul(col[3], 0x0d)  
 c2 = mul(col[0], 0x0d) ^ mul(col[1], 0x09) ^ mul(col[2], 0x0e) ^ mul(col[3], 0x0b)  
 c3 = mul(col[0], 0x0b) ^ mul(col[1], 0x0d) ^ mul(col[2], 0x09) ^ mul(col[3], 0x0e)  
 return [c0, c1, c2, c3]  
  
def inv\_mix\_columns(state):  
 for i in range(4):  
 col = [state[j][i] for j in range(4)]  
 mixed = inv\_mix\_single\_column(col)  
 for j in range(4):  
 state[j][i] = mixed[j]  
 return state  
  
def decrypt\_block(block, expanded\_key):  
 state = [[0]\*4 for \_ in range(4)]  
 for i in range(16):  
 state[i % 4][i // 4] = block[i]  
  
 round\_keys = [expanded\_key[i\*4:(i+1)\*4] for i in range(11)]  
  
 state = add\_round\_key(state, round\_keys[10])  
 state = inv\_shift\_rows(state)  
 state = inv\_sub\_bytes(state)  
  
 for rnd in range(9, 0, -1):  
 state = add\_round\_key(state, round\_keys[rnd])  
 state = inv\_mix\_columns(state)  
 state = inv\_shift\_rows(state)  
 state = inv\_sub\_bytes(state)  
  
 state = add\_round\_key(state, round\_keys[0])  
  
 output = []  
 for i in range(4):  
 for j in range(4):  
 output.append(state[j][i])  
 return bytes(output)