Молдавский Государственный Университет
Факультет Математики и Информатики
Департамент Информатики

Аттестационная работа №2

по курсу “Криптография”

Выполнил: Slavov Constantin,
группа I2302
Проверила: Olga Cerbu, doctor,
conferențiar universitar

Кишинев, 2024

Цель работы: Целью данной работы является изучение алгоритма симметричного блочного шифрования AES (Advanced Encryption Standard) и демонстрация его работы на практике. В рамках исследования необходимо провести шифрование текста с использованием ключа заданной длины, а также ознакомиться с основными этапами алгоритма AES, такими как AddRoundKey, SubBytes, ShiftRows и MixColumns.

В работе также выполняется дешифровка текста с применением обратных операций, включая InvSubBytes, InvShiftRows и InvMixColumns. На каждом этапе преобразования данных будут показаны промежуточные результаты, включая шифрование и восстановление исходного текста.

Задача работы включает оценку надежности и ключевых особенностей алгоритма AES, таких как устойчивость к криптоанализу и сложность обратного восстановления данных без знания ключа. Работа направлена на закрепление теоретических знаний об алгоритмах шифрования и развитие навыков их практического применения.

Содержание: Алгоритм шифрования AES

Незашифрованный текст: CScriptografie24

43 53 63 72 69 70 74 6F 67 72 61 66 69 65 32 34

Ключ: algoritmulAES256

61 6C 67 6F 72 69 74 6D 75 6C 41 45 53 32 35 36

Начальный стейт:

43 53 63 72

69 70 74 6F

67 72 61 66

69 65 32 34

Ключ (W):

61 6C 67 6F

72 69 74 6D

75 6C 4B 45

59 32 35 36

Расширение ключа (Key Expansion) в AES-256 CBC — это процесс, при котором из исходного ключа (master key) создается набор раундовых ключей, необходимых для каждого этапа шифрования и дешифрования. Длина исходного ключа составляет 256 бит, и для работы алгоритма требуется сформировать дополнительные ключи для всех раундов, включая начальный. Трансформация слова (word) включает несколько этапов. Сначала из текущего набора ключевых колонок выбирается последняя колонка. Если она оказывается первой в новом блоке, выполняется циклический сдвиг, при котором первый байт перемещается в конец. Затем к каждому байту применяется замена с использованием S-Box, а первый байт колонки дополнительно подвергается операции XOR с соответствующей константой R-CON.

- Берем последнюю колонку ключа W:

0x53 0x32 0x35 0x36

В бинарном виде: 01010011 00110010 00110101 00110110

- Выполняем циклический сдвиг (circular shift):

Первый байт (0x53) становится последним -

Result: 0x32 0x35 0x36 0x53

В бинарном виде: 00110010 00110101 00110110 01010011

- Выполняем S-Box замену:

Result: 0x23 0x96 0x05 0xE5

В бинарном виде: 00100011 10010110 00000101 11100101

- Выполняем XOR с R-CON:

R-CON для первого раунда: `0x01`. XOR применяется только к первому байту: `23 XOR 01 = 22`

Result: 0x22 0x96 0x05 0xE5

В бинарном виде: 00100010 10010110 00000101 11100101

- Выполняем генерацию нового ключа и группируем его в новую исходную матрицу. Сгенерированная колонка (`22 96 05 E5`) становится новой первой колонкой для следующего раундового ключа. Остальные колонки вычисляются через XOR с предыдущими.

Первая колонка:

`(61 6C 67 6F) XOR (22 96 05 E5) = 43 FA 62 8A`

Вторая колонка:

`(72 69 74 6D) XOR (43 FA 62 8A) = 31 93 16 E7`

Третья колонка:

```
(75 6C 4B 45) XOR (31 93 16 E7) = 44 FF 5D A2
```

Четвёртая колонка:

```
(53 32 35 36) XOR (44 FF 5D A2) = 17 CD 68 94
```

Шаг 4: Итоговый новый ключ

```
0x43 0xFA 0x62 0x8A
0x31 0x93 0x16 0xE7
0x44 0xFF 0x5D 0xA2
0x17 0xCD 0x68 0x94
```

Таким образом, новый сгенерированный ключ представлен как матрица 4x4.

Здесь исходя из байтов 4 колонки ключа был вычислен циклический сдвиг, который используется в алгоритме AES, чтобы создать более сложную зависимость между байтами. Первый байт становится последним.

Далее была выполнена подстановка байтов через S-Box. S-Box - таблица размером 16x16, которая содержит заранее определенные значения. Каждое значение заменяется на новое значение, взятое из этой таблицы. Цель S-Box — усилить запутанность, чтобы сделать шифрование более устойчивым к анализу.

Таблица S-Box значений:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Затем, был выполнен XOR с R-CON. R-CON — это константа для конкретного раунда. Для первого раунда R-CON = `0x01`. XOR применяется только к первому байту. R-CON обеспечивает уникальность каждого раундового ключа.

И, наконец, был сгенерирован новый ключ и итоговая матрица 4x4. Для этого, была использованы данные колонки 0x23 0x96 0x05 0xE5 для создания нового раундового ключа. Остальные колонки формируются с использованием операции XOR между колонками текущего ключа W.

Давайте рассмотрим как это делается на примере формирования первой колонки:

Первая колонка нового ключа формируется как XOR между новой колонкой и первой колонкой предыдущего ключа:

(Первая колонка предыдущего ключа: `61 6C 67 6F`)

(Новая колонка: `22 96 05 E5`)

Рассчитываем XOR для каждого байта:

```
61 XOR 22 = 43
6C XOR 96 = FA
67 XOR 05 = 62
6F XOR E5 = 8A
```

Итог: `43 FA 62 8A`

Таким образом производились все вычисления для 4 колонок и в итоге была получена следующая матрица:

0x43 0xFA 0x62 0x8A

0x31 0x93 0x16 0xE7

0x44 0xFF 0x5D 0xA2

0x17 0xCD 0x68 0x94

Шифрование:

AddRoundKey

В алгоритме шифрования AES операция AddRoundKey является важным этапом, который объединяет текущее состояние данных с ключом раунда. Это обеспечивает внесение запутанности и усиливает защиту сообщения. AddRoundKey применяется на каждом этапе шифрования, включая начальный и завершающий, и представляет собой простую, но высокоэффективную процедуру, где каждый элемент состояния соединяется с соответствующим элементом ключа раунда с помощью операции XOR.

Сообщение:
0x43 0x53 0x63 0x72

0x69 0x70 0x74 0x6F

0x67 0x72 0x61 0x66

0x69 0x65 0x32 0x34

XOR

Ключ:
0x61 0x6C 0x67 0x6F

0x72 0x69 0x74 0x6D

0x75 0x6C 0x4B 0x45

0x53 0x32 0x35 0x36

Result: 0x22 0x3F 0x04 0x1D 0x1B 0x19 0x00 0x02 0x12 0x1E 0x2A 0x23 0x3A 0x57 0x07 0x02

0x22 0x3F 0x04 0x1D

0x1B 0x19 0x00 0x02

0x12 0x1E 0x2A 0x23

0x3A 0x57 0x07 0x02

Round 1:

SubBytes
SubBytes — это операция в алгоритме AES, применяемая на каждом этапе шифрования для внесения запутанности в данные. В ходе этой операции каждый байт текущего состояния заменяется соответствующим значением из специальной таблицы, известной как S-Box.

Начальный стейт:
0x22 0x1B 0x12 0x3A

0x3F 0x19 0x1E 0x57

0x04 0x00 0x2A 0x07

0x1D 0x02 0x23 0x02

Результат после SubBytes:
0x93 0x75 0xF2 0xC1

0xAF 0xD4 0x63 0xC9

0xC9 0x72 0xE5 0x26

0x0A 0x5B 0xC5 0xC9

ShiftRow
ShiftRow — это преобразование в алгоритме AES, направленное на усиление диффузии зашифрованных данных путем изменения порядка байтов в текущем состоянии. Операция выполняется на строках состояния (матрице размером 4x4 байта) и заключается в циклическом сдвиге байтов каждой строки влево, где величина сдвига определяется номером строки.

Начальный стейт:
0x93 0x75 0xF2 0xC1

0xAF 0xD4 0x63 0xC9

0xC9 0x72 0xE5 0x26

0x0A 0x5B 0xC5 0xC9

Правила сдвига строк

Первая строка (0-й сдвиг): остаётся на месте.
`93 75 F2 C1 → 93 75 F2 C1`

Вторая строка (1-й сдвиг): сдвигаем на 1 позицию влево.
`AF D4 63 C9 → D4 63 C9 AF`

Третья строка (2-й сдвиг): сдвигаем на 2 позиции влево.
`C9 72 E5 26 → E5 26 C9 72`

Четвёртая строка (3-й сдвиг): сдвигаем на 3 позиции влево.
`0A 5B C5 C9 → C9 0A 5B C5`

Результат после ShiftRow (сдвигов):
0x93 0xD4 0xE5 0xC9

0x75 0x63 0x26 0x0A

0xF2 0xC9 0xC9 0x5B

0xC1 0xAF 0x72 0xC5

Этап MixColumns в алгоритме AES-128 (режим CBC)

MixColumns — это ключевая операция в алгоритме AES, выполняемая в каждом раунде шифрования, за исключением последнего. Она предназначена для перемешивания данных в столбцах состояния, что усиливает диффузию и обеспечивает взаимное влияние всех байтов в колонке.

Состояние в алгоритме AES представляется как матрица размером 4×4, где каждый элемент является байтом, рассматриваемым как элемент конечного поля GF(28)GF(2^8). Операция MixColumns заключается в линейном преобразовании, где каждый столбец состояния умножается на фиксированную матрицу MM размером 4×4. В результате выполняется вычисление нового столбца с использованием операций умножения и сложения в конечном поле GF(28)GF(2^8).

Все арифметические действия выполняются с учетом свойств конечного поля. Умножение на 1 оставляет байт без изменений. Умножение на 2 реализуется сдвигом байта влево на один бит с последующим возможным применением операции XOR с неприводимым многочленом x8+x4+x3+x+1x^8 + x^4 + x^3 + x + 1 (в шестнадцатеричной форме 0x1B0x1B). Умножение на 3 вычисляется как XOR результата умножения на 2 с исходным значением байта. Например, для байта s=0x57s = 0x57 (в

двоичном виде 0101011101010111) умножение на 2 выполняется сдвигом 01010111→10101110 \to 10101110, а затем, при необходимости, применяется XOR с 0x1B0x1B. Умножение на 3 вычисляется как результат XOR (s⋅2)⊕s(s \cdot 2) \oplus s.

MixColumns обеспечивает эффективное перемешивание байтов внутри колонок, создавая более сложные зависимости между данными, что существенно повышает криптографическую стойкость алгоритма.

Давайте рассмотрим все эти вычисления на примере:

Для этого этапа используем ваши данные состояния **после ShiftRows** и применяем фиксированную матрицу M:

Исходная матрица:
0x93 0xD4 0xE5 0xC9

0x75 0x63 0x26 0x0A

0xF2 0xC9 0xC9 0x5B

0xC1 0xAF 0x72 0xC5

Матрица M:
02 03 01 01

01 02 03 01

01 01 02 03

03 01 01 02

MixColumns применяется к каждой колонке состояния отдельно.
- Колонка 1:
0x93 0xD4 0xE5 0xC9

Используем формулу для вычисления строки 1 колонки C':

C'[0] = (02 * 93) ⊕ (03 * 75) ⊕ (01 * F2) ⊕ (01 * C1)

Аналогично вычисляем остальные строки для этой колонки.

Таким образом производятся вычисления для всех колонок, после чего создается новая матрица.

Зачем нужен MixColumns?
Операция MixColumns усиливает диффузию данных, обеспечивая распространение изменений одного байта входных данных на всю колонку. Это значительно повышает стойкость шифрования к криптоанализу, так как взаимосвязь между байтами увеличивается с каждым раундом шифрования.

Особенности в режиме CBC заключаются в том, что к каждому входному блоку добавляется результат операции XOR с предыдущим зашифрованным блоком (или вектором инициализации для первого блока). Однако сам MixColumns остается неизменной частью раундовых преобразований, выполняя свою задачу независимо от режима работы.

Результат этапа MixColumns:
0x91 0x70 0x00 0x09

0xB5 0xFD 0x9B 0xF5

0x41 0xD4 0xDC 0x21

0xB0 0x88 0x3F 0x80

Объяснение вычислений:

Для каждой колонки исходного состояния применена фиксированная матрица `M`:

```
02 03 01 01

01 02 03 01

01 01 02 03

03 01 01 02
```

1. Умножения выполнялись в конечном поле GF(2^8), с использованием неприводимого многочлена AES (`x^8 + x^4 + x^3 + x + 1 = 0x1B`).
2. Итоговые значения получены сложением XOR всех произведений.

Этап AddRoundKey

Что такое AddRoundKey?

AddRoundKey — это операция в алгоритме AES, где каждое состояние (матрица State) комбинируется с ключом текущего раунда с использованием операции **XOR**. Цель — добавить компонент раундового ключа к состоянию для усиления стойкости шифрования.

Начальный стейт:
0x91 0x70 0x00 0x09

0xB5 0xFD 0x9B 0xF5

0x41 0xD4 0xDC 0x21

0xB0 0x88 0x3F 0x80

Давайте рассмотрим работу AddRoundKey на примере первой колонки:

Операция XOR выполняется поэлементно между каждым байтом состояния и соответствующим байтом ключа:

Первая колонка:

```
91 XOR A1 = 30
B5 XOR E5 = 50
41 XOR 09 = 48
B0 XOR 13 = A3
```

Результат первой колонки:

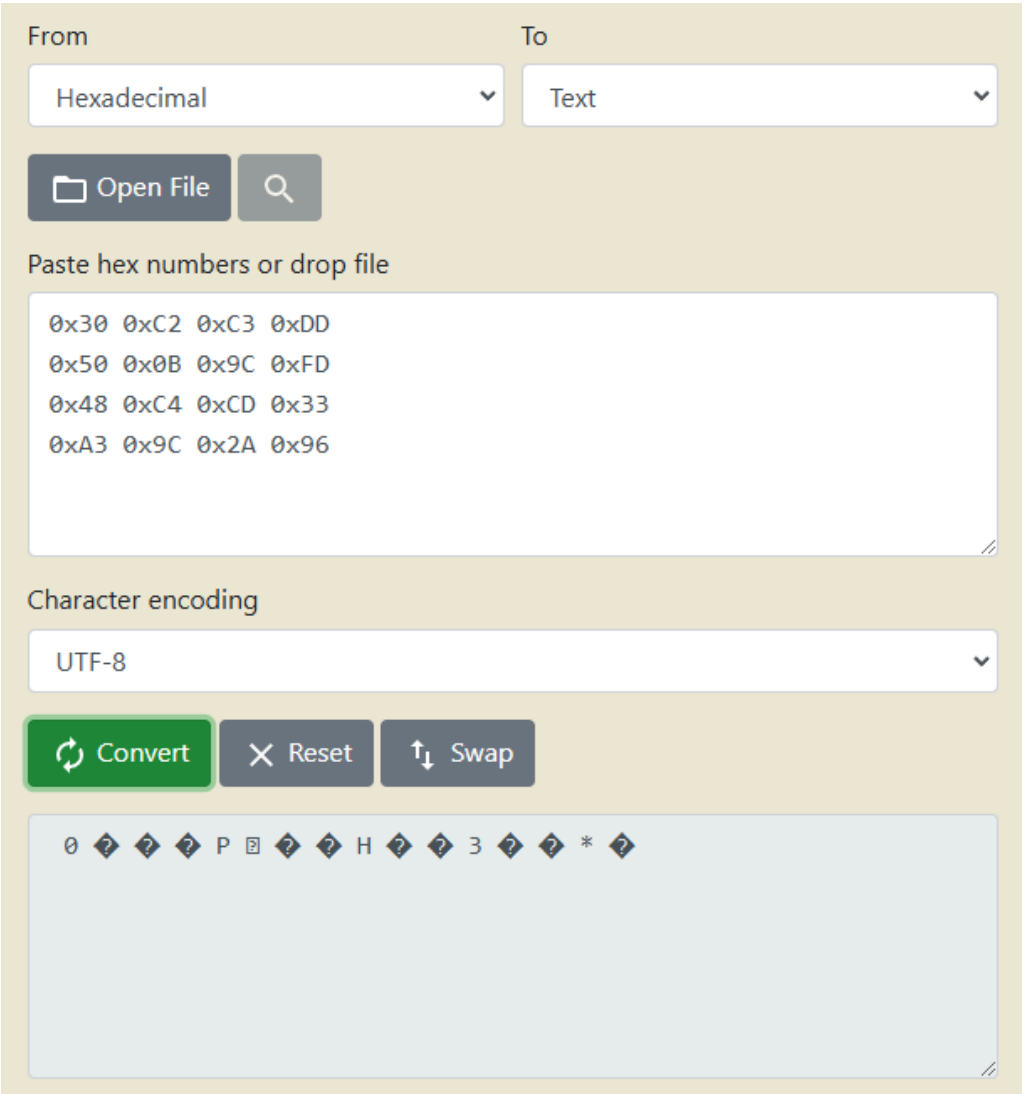
```
30 50 48 A3
```

Результат после AddRoundKey:
0x30 0xC2 0xC3 0xDD

0x50 0x0B 0x9C 0xFD

0x48 0xC4 0xCD 0x33

0xA3 0x9C 0x2A 0x96



- Дешифровка -

Дешифровка — это обратный процесс, в котором применяются AddRoundKey, InvMixColumn, InvShiftRow и InvByteSub для получения исходного сообщения.

Исходное сообщение:

0x30 0xC2 0xC3 0xDD 0x50 0x0B 0x9C 0xFD 0x48 0xC4 0xCD 0x33 0xA3 0x9C 0x2A 0x96

Перевод в бинарный код:

00110000 11000010 11000011 11011101 01010000 00001011 10011100 11111101 01001000 11000100 11001101 00110011 10100011 10011100 00101010 10010110

Состояние после AddRoundKey:

0x51 0xAE 0xA4 0xB2 0x22 0x62 0xE8 0x90 0x3D 0xA8 0x8C 0x76 0xF0 0xAE 0x1F 0xA0

Перевод в бинарный код:

01010001 10101110 10100100 10110010 00100010 01100010 11101000 10010000 00111101 10101000 10001100 01110110 11110000 10101110 00011111 10100000

Состояние после InvShiftRows:

0x51 0x90 0x8C 0xA0 0x22 0x62 0x3D 0xAE 0xE8 0xB2 0xA4 0x76 0xF0 0x1F 0xAE 0xA8

Перевод в бинарный код:

01010001 10010000 10001100 10100000 00100010 01100010 00111101 10101110 11101000 10110010 10100100 01110110 11110000 00011111 10101110 10101000

Состояние после InvSubBytes:

0x30 0xC2 0xC3 0xDD 0x50 0x0B 0x9C 0xFD 0x48 0xC4 0xCD 0x33 0xA3 0x9C 0x2A 0x96

Перевод в бинарный код:

00110000 11000010 11000011 11011101 01010000 00001011 10011100 11111101 01001000 11000100 11001101 00110011 10100011 10011100 00101010 10010110

- Round 1 -

Состояние после AddRoundKey (Round 1):

0x55 0x06 0xA3 0x8A 0xFD 0x22 0x4A 0x21 0xB7 0x7A 0xBF 0x50 0x18 0xD9 0x47 0xFC

Перевод в бинарный код:

01010101 00000110 10100011 10001010 11111101 00100010 01001010 00100001 10110111 01111010 10111111 01010000 00011000 11011001 01000111 11111100

Состояние после InvMixColumns:

Для каждой колонки выполняется отдельно вычисление.

- Первая колонка:

a[0,0] = (0x0E * 0x81) ⊕ (0x0B * 0xD1) ⊕ (0x0D * 0x3F) ⊕ (0x09 * 0x61) = 0x93;

a[1,0] = (0x09 * 0x81) ⊕ (0x0E * 0xD1) ⊕ (0x0B * 0x3F) ⊕ (0x0D * 0x61) = 0xD4;

a[2,0] = (0x0D * 0x81) ⊕ (0x09 * 0xD1) ⊕ (0x0E * 0x3F) ⊕ (0x0B * 0x61) = 0xE5;

a[3,0] = (0x0B * 0x81) ⊕ (0x0D * 0xD1) ⊕ (0x09 * 0x3F) ⊕ (0x0E * 0x61) = 0xC9;

- Вторая колонка:

a[0,1] = (0x0E * 0xD6) ⊕ (0x0B * 0x9D) ⊕ (0x0D * 0x6A) ⊕ (0x09 * 0xED) = 0x75;

a[1,1] = (0x09 * 0xD6) ⊕ (0x0E * 0x9D) ⊕ (0x0B * 0x6A) ⊕ (0x0D * 0xED) = 0x63;

a[2,1] = (0x0D * 0xD6) ⊕ (0x09 * 0x9D) ⊕ (0x0E * 0x6A) ⊕ (0x0B * 0xED) = 0x26;

a[3,1] = (0x0B * 0xD6) ⊕ (0x0D * 0x9D) ⊕ (0x09 * 0x6A) ⊕ (0x0E * 0xED) = 0x0A;

- Третья колонка:

a[0,2] = (0x0E * 0x83) ⊕ (0x0B * 0x15) ⊕ (0x0D * 0x73) ⊕ (0x09 * 0xCA) = 0xF2;

a[1,2] = (0x09 * 0x83) ⊕ (0x0E * 0x15) ⊕ (0x0B * 0x73) ⊕ (0x0D * 0xCA) = 0xC9;

a[2,2] = (0x0D * 0x83) ⊕ (0x09 * 0x15) ⊕ (0x0E * 0x73) ⊕ (0x0B * 0xCA) = 0xC9;

a[3,2] = (0x0B * 0x83) ⊕ (0x0D * 0x15) ⊕ (0x09 * 0x73) ⊕ (0x0E * 0xCA) = 0x5B;

- Четвёртая колонка:

a[0,3] = (0x0E * 0xAE) ⊕ (0x0B * 0xED) ⊕ (0x0D * 0x04) ⊕ (0x09 * 0x3C) = 0xC1;

a[1,3] = (0x09 * 0xAE) ⊕ (0x0E * 0xED) ⊕ (0x0B * 0x04) ⊕ (0x0D * 0x3C) = 0xAF;

a[2,3] = (0x0D * 0xAE) ⊕ (0x09 * 0xED) ⊕ (0x0E * 0x04) ⊕ (0x0B * 0x3C) = 0x72;

a[3,3] = (0x0B * 0xAE) ⊕ (0x0D * 0xED) ⊕ (0x09 * 0x04) ⊕ (0x0E * 0x3C) = 0xC5;

Итог: 0x81 0xD6 0x83 0xAE 0xD1 0x9D 0x15 0xED 0x3F 0x6A 0x73 0x04 0x61 0xED 0xCA 0x3C

Перевод в бинарный код:

10010011 11010100 11100101 11001001 01110101 01100011 00100110 00001010 11110010 11001001 11001001 01011011 11000001 10101111 01110010 11000101

Состояние после InvShiftRows:

InvShiftRow — это операция, используемая в процессе расшифровки алгоритма AES и являющаяся обратной операцией к преобразованию ShiftRow, которое применяется при шифровании. В этой операции каждая строка состояния сдвигается вправо на разное количество позиций в зависимости от её номера:

Сдвиг строк вправо:

1. Первая строка остаётся неизменной.
2. Вторая строка сдвигается вправо на 1 байт.
3. Третья строка сдвигается вправо на 2 байта.
4. Четвёртая строка сдвигается вправо на 3 байта.

0x93 0x96 0x0E 0x14 0xCB 0x4C 0xEC 0x16 0x20 0x9F 0x75 0x2D 0x03 0xAB 0x52 0xCC

Перевод в бинарный код:

10000001 11010110 10000011 10101110 11010001 10011101 00010101 11101101 00111111 01101010 01110011 00000100 01100001 11101101 11001010 00111100

Состояние после InvSubBytes:

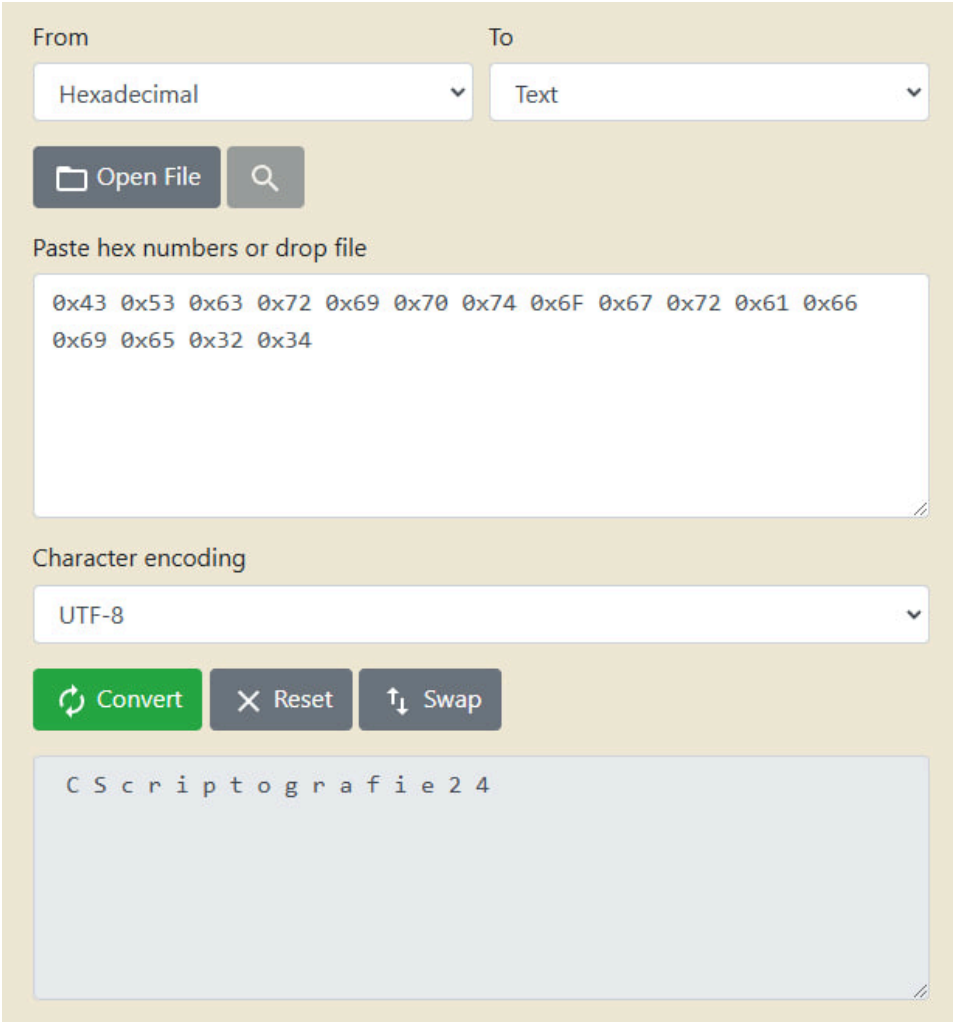
0x22 0x35 0xD7 0x9B 0x59 0x5D 0x83 0xFF 0x54 0x6E 0x3F 0xFA 0xD5 0x0E 0x48 0x27

Перевод в бинарный код:

00100010 00110101 11010111 10011011 01011001 01011101 10000011 11111111 01010100 01101110 00111111 11111010 11010101 00001110 01001000 00100111

AddRoundKey (последний шаг)

В процессе расшифровки AES операция AddRoundKey выполняет ту же функцию, что и при шифровании, но в обратном порядке. Она заключается в том, чтобы объединить текущее состояние с ключом соответствующего раунда. Однако в отличие от шифрования, на этапе расшифровки этот процесс начинается с последнего раунда и идет к первому.



Вывод: Полученный результат — успешно расшифрованное сообщение "CScriptografie24", которое полностью соответствует оригинальному тексту. Это демонстрирует корректную работу алгоритма AES и использование заданного ключа, обеспечив точное шифрование и последующую расшифровку данных.

Версия AES-128 отличается длиной ключа, которая составляет 128 бит. В отличие от более длинных вариантов, таких как AES-192 или AES-256, более короткий ключ обеспечивает более высокую скорость выполнения алгоритма, однако немного уступает в уровне теоретической безопасности.

Режим CBC (Cipher Block Chaining) добавляет зависимость каждого шифруемого блока данных от предыдущего. Это позволяет избежать повторяющихся паттернов в зашифрованных данных, усиливая безопасность алгоритма и защищая от атак, основанных на анализе повторений.

Часть 2: Подпись Меркла

Подпись Меркла — это криптографический метод создания цифровых подписей, который был изобретён Ральфом Мерклом в 1979 году. Она основывается на хэш-функциях и используется для защиты больших объёмов данных. Вместо того чтобы подписывать каждое сообщение отдельно, этот метод позволяет подписать множество данных, создав единую подпись.

- Как работает подпись Меркла?

***Хэширование данных**:*

- Вначале данные (или сообщение) преобразуются в хэши. Хэш — это уникальный "отпечаток", который позволяет проверить, что данные не были изменены.

- Чтобы создать **хэш** слова "Kostya", используется специальный математический алгоритм, называемый **хэш-функцией**. Она преобразует любые данные (например, текст или файл) в уникальный код фиксированной длины.

Вот как это работает:

- Выбор хэш-функции:

- Например, возьмём одну из самых популярных хэш-функций — **SHA-256**. Она используется во многих современных системах безопасности.

- Эта функция принимает любое слово, число или текст, а возвращает строку из 64 символов (в шестнадцатеричном формате).

Обработка слова "Kostya":

- Вы вводите слово "Kostya" в хэш-функцию.

- Алгоритм разбивает текст на части, кодирует их числами и проводит множество математических операций.

- Результатом становится уникальная последовательность символов, которая представляет это слово.

Хэш для слова **"Kostya"** с использованием алгоритма SHA-256:
bfaf8f153d89a42fcd6086e8980f66fd49416005e76f8d3e992e8dcb64ef2d80

Что важно знать:

- Уникальность: Даже если вы измените хотя бы одну букву (например, напишете "Kostya" с заглавной буквы), хэш изменится полностью.

- Длина: Независимо от того, что вы вводите — слово, предложение или целую книгу — результат всегда одной длины (для SHA-256 это 64 символа).

- ***Дерево Меркла****:*

Все хэши данных объединяются в бинарное дерево (дерево с двумя "ветвями" на каждом уровне).

Хэши пар объединяются и снова хэшируются, чтобы сформировать новый уровень дерева.

Вершина дерева — **корень Меркла** (Merkle Root) — это единое значение, которое представляет все данные в дереве.

- Подпись корня дерева:

Вместо того чтобы подписывать каждое сообщение или хэш, подписывается только корень дерева. Это позволяет сэкономить ресурсы.

- Проверка подписи:

Для проверки подписи сравнивается корень дерева с хэшем подписанного значения.

Если корень совпадает, то данные считаются подлинными.

- Преимущества и недостатки

Преимущества:

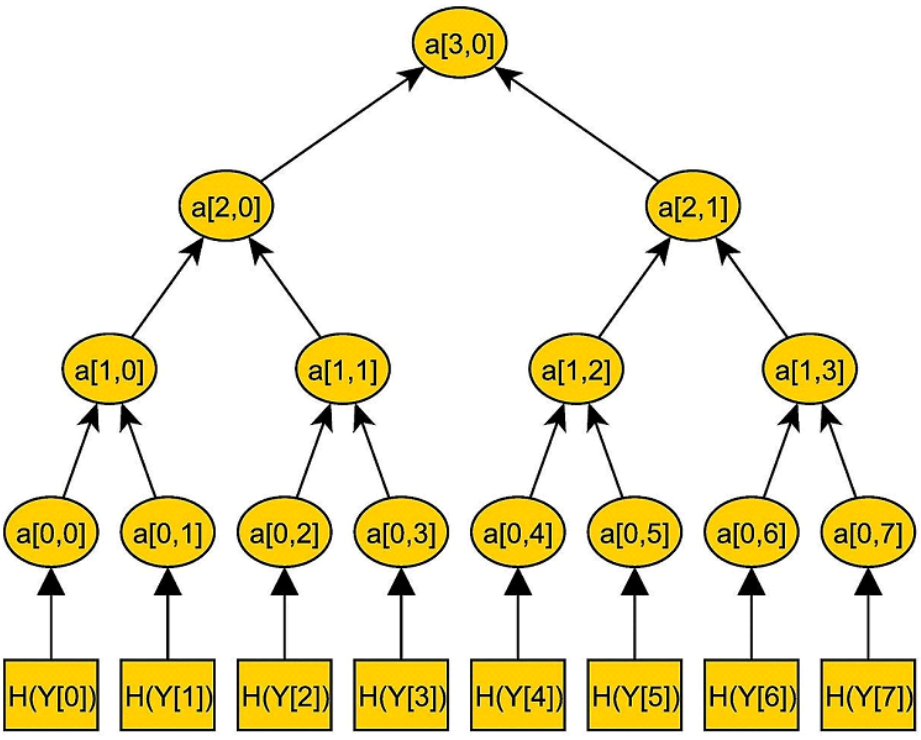
- Эффективность: Подпись создаётся быстро, даже для больших объёмов данных.

- Надёжность: Использование хэш-функций делает метод устойчивым к взлому.

Недостатки:

- Реализация может быть сложнее, чем у стандартных схем подписей, таких как RSA.

- Требует предварительного построения дерева, что занимает ресурсы.



- Пример создания подписи Меркла

Генерация ключей

- Приватные ключи (секретные данные):

Пример:

X = ["Key1", "Key2", "Key3", "Key4"].

-Публичные ключи (хэши секретных данных):

Y = [H(Key1), H(Key2), H(Key3), H(Key4)].

- Построение дерева

1. На первом уровне (листья дерева) находятся хэши:
   a_0 = [H(Key1), H(Key2), H(Key3), H(Key4)].
2. Далее объединяем хэши парами и снова хэшируем:
   a_{1,0} = H(H(Key1) || H(Key2))
   a_{1,1} = H(H(Key3) || H(Key4))
3. На последнем уровне объединяем хэши, чтобы создать корень дерева:
   a_2 = H(a_{1,0} || a_{1,1})

Корень дерева (a_2) — это единая подпись для всех данных.

- Генерация подписи

Для подписания сообщения выбирается приватный ключ, например, Key1.

Создаётся подпись, включающая хэш сообщения и путь в дереве от листа до корня.

- Проверка подписи

Проверка начинается с хэша листа (хэш данных).

По цепочке (аутентификационному пути) хэши соединяются до уровня корня.

Если полученный корень совпадает с подписанным, подпись считается действительной.

- Пример с числами

Допустим, у нас есть 4 ключа:
Key1 = "SC1", Key2 = "SC2", Key3 = "SC3", Key4 = "SC4"

- Построение дерева:

Хэшируем ключи:
H(Key1), H(Key2), H(Key3), H(Key4)

Объединяем и хешируем:
H(H(Key1) || H(Key2)), H(H(Key3) || H(Key4))

Создаём корень, он и станет подписью:
H(H(H(Key1) || H(Key2)) || H(H(Key3) || H(Key4)))

- Почему это удобно?

Быстро и экономично:
Вы подписываете не каждый файл, а сразу весь набор данных.

Безопасно:
Даже если кто-то изменит хоть один файл, корень дерева (ваша подпись) изменится.

Просто проверить:

Для проверки достаточно знать только корень дерева и несколько промежуточных хэшей.

- Где применяется подпись Меркла?

Блокчейн:

Например, в Bitcoin дерево Меркла помогает проверять транзакции в блоках.

Системы хранения данных:

Чтобы убедиться, что файлы не были подменены.

Сетевые приложения:

Проверка подлинности данных в распределённых системах.

- Итог:

Подпись Меркла — это эффективный способ работы с большими данными, который сочетает безопасность хэш-функций и удобство структуры дерева. Вместо того чтобы подписывать каждый элемент данных отдельно, подписывается только корень дерева, что экономит ресурсы и ускоряет проверку.

Метод позволяет быстро обнаружить любые изменения в данных: если хотя бы один элемент будет изменён, корень дерева станет другим, и подпись потеряет свою силу. Это делает подпись Меркла надёжной и стойкой к подмене.

Подпись широко используется в блокчейне для проверки транзакций, в системах хранения для защиты файлов и в сетевых протоколах для проверки данных. Это универсальный инструмент, который упрощает работу с большими объёмами информации и делает её безопасной.