МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ

ДЕПАРТАМЕНТ ИНФОРМАТИКИ

“Криптография и безопасность информации”

**Аттестация №2**

Tema: Algoritmul de criptare/decriptare AES  
**Mercle Signature**

**Автор:** Калинкова С.

группа I2302

**Проверила:** Чербу О.

Кишинев, 2024

**Sarcina A) algoritmul AES**

**AES (Advanced Encryption Standard)** — это симметричный алгоритм шифрования, который используется для защиты данных. Он основан на блочном шифровании и работает с блоками данных размером 128 бит. Ключи могут иметь длину 128, 192 или 256 бит, что определяет уровень безопасности.

### **Основные особенности:**

* **Тип шифрования**: Симметричное (тот же ключ для шифрования и дешифрования).
* **Алгоритм**: Работает по принципу замены и перестановки данных через несколько раундов (10, 12 или 14 раундов в зависимости от длины ключа).
* **Процесс**: Каждое слово блока данных шифруется через несколько этапов: подстановка, перестановка, добавление ключа и сдвиг строк.

**AES работает следующим образом:**

1. **Инициализация**: Блок данных (128 бит) и ключ (128, 192 или 256 бит) разделяются на подблоки.
2. **Раунды шифрования**:
   * **SubBytes**: Каждый байт блока заменяется с помощью таблицы подстановок (S-Box).
   * **ShiftRows**: Строки блока сдвигаются циклично.
   * **MixColumns**: Каждый столбец блока изменяется с помощью линейной перестановки.
   * **AddRoundKey**: Блок данных комбинируется с ключом с помощью побитовой операции XOR.
3. **Последний раунд**: Без этапа **MixColumns**.
4. **Дешифрование**: Процесс обратный шифрованию, но с инвертированными операциями.

Шифрование происходит в несколько раундов, в зависимости от длины ключа (10, 12 или 14 раундов).

AES широко используется для защиты данных в различных областях, включая интернет-безопасность и шифрование файлов.

**Ход работы:**

Условие: Зашифровать и расшифровать 128-битное сообщение (строго 16 символов), сообщение будет начинаться с инициалов студента, остальные данные выбираются по вашему выбору. (Выполните все операции, разработайте программу, которая зашифрует/дешифрует выбранные данные)

Начальный текс 128 бит:

Сообщение: CS20streamlined9  
Для начала, мы преобразуем каждую букву и цифру строки в их соответствующие байты в ASCII.

| 43 | 53 | 32 | 30 | 73 | 74 | 72 | 65 | 61 | 6D | 6C | 69 | 6E | 65 | 64 | 39 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Cheia: thunderstorm0123 (128 biti):

| 74 | 68 | 75 | 6E | 64 | 65 | 72 | 73 | 74 | 6F | 72 | 6D | 30 | 31 | 32 | 33 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Nb = 128/32 = 4 число колонок)

Nk = 128/32 = 4 (число слов в исходном ключе)

Nr = 10 (количество раундов)

Starea Initiala

| 43 | 53 | 32 | 30 |
| --- | --- | --- | --- |
| 73 | 74 | 72 | 65 |
| 61 | 6D | 6C | 69 |
| 6E | 65 | 64 | 39 |

Cheia (W):

| ***0*** | 74 | 68 | 75 | 6E |
| --- | --- | --- | --- | --- |
| ***1*** | 64 | 65 | 72 | 73 |
| ***2*** | 74 | 6F | 72 | 6D |
| ***3*** | 30 | 31 | 32 | 33 |
| ***4*** | … | | | |

**Ключевое расширение (Key Expansion)**

**Расширенный ключ** в алгоритме AES — это набор ключей, которые генерируются на основе начального ключа и используются на каждом этапе шифрования. Используется для:

1. **Защита данных:** На каждом раунде алгоритма AES используется свой уникальный раундовый ключ, полученный из расширенного ключа. Это увеличивает устойчивость алгоритма к атакам.
2. **Ускорение вычислений:** Расширенный ключ вычисляется заранее (до шифрования или расшифрования), что позволяет избежать дополнительных затрат времени на его генерацию во время работы.

Для преобразований используем RotWord и SubWord:

1)row=Nk=4 (это индекс строки)

temp = W[row-1]=W[4-1]=W[3] = 30 31 32 33 (значение текущего слова в расширении ключа)

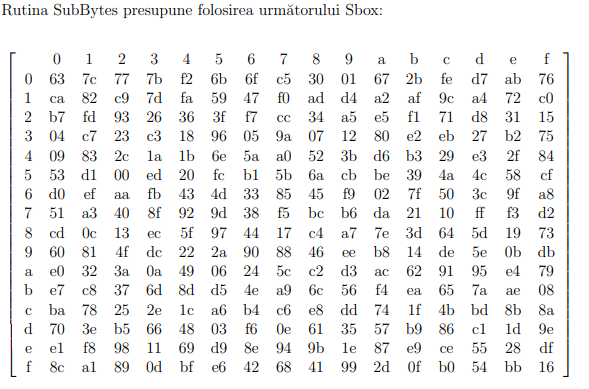
row%Nk=4%4=0 =>

temp = SubWord(RotWord(temp))

RotWord(temp) = 30 31 32 33 <<<1 = 31 32 33 30

SubWord(RotWord(temp)) = SubWord(31 32 33 30)  
В этом фрагменте описан процесс расширения ключа. Для текущего слова ключа W[3] = 30 31 32 33 выполняются две операции: сначала циклический сдвиг (RotWord), который перемещает байты влево, результатом чего становится 31 32 33 30, а затем каждый байт проходит через операцию подстановки (SubWord), заменяя его на соответствующее значение из таблицы подстановки (S-box).

Используем таблицу замены:



31 ->c7 , 32 -> 23 , 33 -> c3, 30 -> 04

Primim temp=SubWord(RotWord(temp)) = c7 23 c3 04

temp = temp XOR Rcon[row/Nk] = temp XOR Rcon[4/4] = temp XOR Rcon[1] = c7 23 c3 04 xor 01000000 = **c6 23 c3 44  
пояснение** После этого к полученному результату применяется операция XOR с константой раунда Rcon[1] = 01 00 00 00, что дает итоговое значение c6 23 c3 44. Это новое слово ключа используется в дальнейших раундах шифрования.

1. c7 XOR 01 = c6
2. 23 XOR 00 = 23
3. c3 XOR 00 = c3
4. 04 XOR 40 = 44

W[4] = W[row-Nk] xor temp = W[4-4] xor temp = W[0] xor temp = 7468756E xor **c6 23 c3 44** = **b2 4b b6 2a**

**пояснение** Для вычисления W[4] выполняется операция XOR между первым словом ключа и результатом предыдущих операций. Результат становится новым значением для W[4].

1. 74 XOR c6 = b2
2. 68 XOR 23 = 4b
3. 75 XOR c3 = b6
4. 6E XOR 44 = 2a

W:

| ***0*** | 74 | 68 | 75 | 6E |
| --- | --- | --- | --- | --- |
| ***1*** | 64 | 65 | 72 | 73 |
| ***2*** | 74 | 6F | 72 | 6D |
| ***3*** | 30 | 31 | 32 | 33 |
| ***4*** | b2 | 4b | b6 | 2a |

2)row=row+1=4+1=5

temp = W[row-1]=W[5-1]=W[4] = B2 4B B6 2A

row%Nk=5%4=1, Nk<>8, row%Nk<>4 =>

W[5] = W[5- Nk] XOR Temp = W[5-4] xor temp = 64 65 72 73 XOR B2 4B B6 2A = **D6 2E C4 59   
64 XOR B2 = D6**

**65 XOR 4B = 2E**

**72 XOR B6 = C4**

**73 XOR 2A = 59**W:

| ***0*** | 74 | 68 | 75 | 6E |
| --- | --- | --- | --- | --- |
| ***1*** | 64 | 65 | 72 | 73 |
| ***2*** | 74 | 6F | 72 | 6D |
| ***3*** | 30 | 31 | 32 | 33 |
| ***4*** | b2 | 4b | b6 | 2a |
| ***5*** | D6 | 2E | C4 | 59 |

3)row=row+1=5+1=6

temp = W[row-1]=W[6-1]=W[5] = **D6 2E C4 59**

row% Nk =6%4=2≠0 =>

W[6] = W[6- Nk] XOR temp = W[2] XOR temp = 746F726D XOR D62EC459 =  **A241B634**

W:

| ***0*** | 74 | 68 | 75 | 6E |
| --- | --- | --- | --- | --- |
| ***1*** | 64 | 65 | 72 | 73 |
| ***2*** | 74 | 6F | 72 | 6D |
| ***3*** | 30 | 31 | 32 | 33 |
| ***4*** | b2 | 4b | b6 | 2a |
| ***5*** | D6 | 2E | C4 | 59 |
| ***6*** | A2 | 41 | B6 | 34 |

4)row=row+1=6+1=7

row% Nk = 7%4=3≠0

W[7] = W[7- Nk] XOR temp = W[3] XOR temp = 30313233 XOR **A241B634** = **92 70 84 07**

W:

| ***0*** | 74 | 68 | 75 | 6E |
| --- | --- | --- | --- | --- |
| ***1*** | 64 | 65 | 72 | 73 |
| ***2*** | 74 | 6F | 72 | 6D |
| ***3*** | 30 | 31 | 32 | 33 |
| ***4*** | b2 | 4b | b6 | 2a |
| ***5*** | D6 | 2E | C4 | 59 |
| ***6*** | A2 | 41 | B6 | 34 |
| ***7*** | 92 | 70 | 84 | 07 |

**Шифрование**

**AddRoundKey:**

В шифровании AES операция AddRoundKey необходима для объединения текущего состояния данных с раундовым ключом, что способствует запутыванию и безопасности сообщений. AddRoundKey применяется к каждому раунду шифрования, включая первый и последний, и состоит из простой, но очень эффективной операции:

Каждый элемент состояния подвергается операции XOR с соответствующим ключом для текущего раунда

*Stare[i,j]= Stare[i,j] XOR W[round\*4+j,i]*

round = 0

Starea initiala:

| 43 | 53 | 32 | 30 |
| --- | --- | --- | --- |
| 73 | 74 | 72 | 65 |
| 61 | 6D | 6C | 69 |
| 6E | 65 | 64 | 39 |

XOR

Cheia utilizata:

| 74 | 68 | 75 | 6E |
| --- | --- | --- | --- |
| 64 | 65 | 72 | 73 |
| 74 | 6F | 72 | 6D |
| 30 | 31 | 32 | 33 |

**Первая строка:**

* 43 XOR 74 = 37
* 53 XOR 68 = 3B
* 32 XOR 75 = 47
* 30 XOR 6E = 5E

**Вторая строка:**

* 73 XOR 64 = 17
* 74 XOR 65 = 11
* 72 XOR 72 = 00
* 65 XOR 73 = 16

**Третья строка:**

* 61 XOR 74 = 15
* 6D XOR 6F = 02
* 6C XOR 72 = 1E
* 69 XOR 6D = 04

**Четвёртая строка:**

* 6E XOR 30 = 5E
* 65 XOR 31 = 54
* 64 XOR 32 = 56
* 39 XOR 33 = 0A

**Результат:** 37 3B 47 5E 17 11 00 16 15 02 1E 04 5E 54 56 0A

| 37 | 3B | 47 | 5E |
| --- | --- | --- | --- |
| 17 | 11 | 00 | 16 |
| 15 | 02 | 1E | 04 |
| 1A | 5E | 54 | 0A |

**Runda 1:**

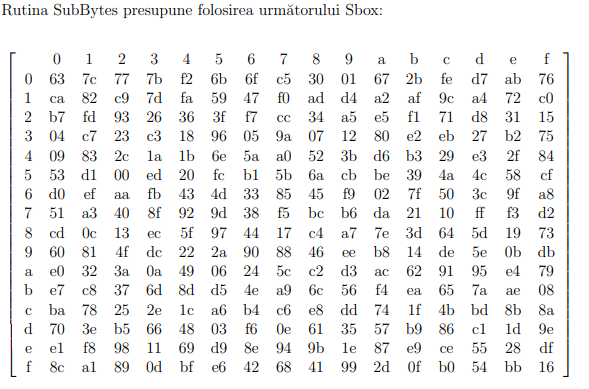
**ByteSub:**(Заменяет каждое значение согласно таблице подстановки)

ByteSub — это эффективная операция алгоритма AES, применяемая на каждом этапе шифрования для внесения путаницы в зашифрованные данные. Внутри ByteSub каждый байт в текущем состоянии заменяется определенным значением из таблицы S-Box.

Starea Initiala:

| 37 | 3B | 47 | 5E |
| --- | --- | --- | --- |
| 17 | 11 | 00 | 16 |
| 15 | 02 | 1E | 04 |
| 1A | 5E | 54 | 0A |

Folosim tabelul de substitutie.



Starea obtinuta:

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| f0 | 82 | 63 | 47 |
| 59 | 77 | 72 | f2 |
| A2 | 58 | 20 | 67 |

**ShiftRow:** (сдвигайте строки для увеличения распространения)

ShiftRow — это операция преобразования, используемая в алгоритме AES для увеличения распространения зашифрованных данных путем изменения порядка байтов в текущем состоянии. Эта операция применяется к строкам состояния (матрица 4x4 байта) и состоит из кругового сдвига влево от каждой строки, различного в зависимости от номера строки.

Как работает ShiftRow:

* Ряд 0: оставить без изменений.
* Ряд 1: сдвигается на одну позицию влево.
* Ряд 2: сдвигается на две позиции влево.
* Ряд 3: сдвигается на три позиции влево.

Starea Initiala:

| 9A | e2 | A0 | 58 | <<<0 |
| --- | --- | --- | --- | --- |
| f0 | 82 | 63 | 47 | <<<1 |
| 59 | 77 | 72 | f2 | <<<2 |
| A2 | 58 | 20 | 67 | <<<3 |

Starea obtinuta: (hexazecimal)

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| 82 | 63 | 47 | f0 |
| 72 | f2 | 59 | 77 |
| 67 | A2 | 58 | 20 |

Starea obtinuta: (zecimal)

| 154 | 226 | 160 | 88 |
| --- | --- | --- | --- |
| 130 | 99 | 71 | 240 |
| 114 | 242 | 89 | 119 |
| 103 | 162 | 88 | 32 |

**MixColumn: Преобразование на основе операций умножения Галуа, что увеличивает путаницу.**

(Расчет для каждого элемента b[i, j] столбцов)

Построение таблиц 2\*а и 3\*а

2а и 3а представляют собой значения, полученные умножением каждого элемента в текущем состоянии на 2 и 3 соответственно.

Таблица 2\*а содержит значения, полученные в результате умножения каждого элемента столбцов на 2, а таблица 3\*а — значения, умноженные на 3.

Crearea tabelului:

2\*a:  
 2\*

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| 82 | 63 | 47 | f0 |
| 72 | f2 | 59 | 77 |
| 67 | A2 | 58 | 20 |

**Основные шаги:**

1. Преобразуем значение в **двоичную форму**.
2. Умножаем на 2 (что эквивалентно сдвигу на 1 бит влево).
3. Если результат превышает 255 (или 0xFF в шестнадцатеричной системе), применяем **XOR с маской 0x1B**, чтобы "сжать" результат в пределах одного байта.

| **Primul rând**  9A:  Binar: 10011010  Înmulțit cu 2: 00110100 = 34 | **Al doilea rând**  82:  Binar: 10000010  Înmulțit cu 2: 00000100  = 04 | **Al treilea rând**  72:  Binar: 01110010  Înmulțit cu 2: 11100100= E4 | **Al patrulea rând**  67:  Binar: 01100111  Înmulțit cu 2: 11001110 = CE |
| --- | --- | --- | --- |
| E2:  Binar: 11100010  Înmulțit cu 2: 11000100 XOR with 1B = DF | 63:  Binar: 01100011  Înmulțit cu 2: 11000110 = C6 | F2:  Binar: 11110010  Înmulțit cu 2: 111100100 XOR with 1B = E5 | A2:  Binar: 10100010  Înmulțit cu 2: 101000100 XOR with 1B = B7 |
| A0:  Binar: 10100000  Înmulțit cu 2: 01000000  = 40 | 47:  Binar: 01000111  Înmulțit cu 2: 10001110= 8E | 59:  Binar: 01011001  Înmulțit cu 2: 10110010  = B2 | 58:  Binar: 01011000  Înmulțit cu 2: 10110000= B0 |
| 58:  Binar: 01011000  Înmulțit cu 2: 10110000= B0 | F0:  Binar: 11110000  Înmulțit cu 2: 111100000 XOR with 1B = EB | 77:  Binar: 01110111  Înmulțit cu 2: 11101110= EE | 20:  Binar: 00100000  Înmulțit cu 2: 1000000  40 |

Итоговая таблица после умножения на 2

| 34 | DF | 40 | B0 |
| --- | --- | --- | --- |
| 04 | C6 | 8E | EB |
| E4 | E5 | B2 | EE |
| CE | B7 | B0 | 40 |

3\*a:

В AES умножение на 3 — это не простая операция удвоения (умножение на 2). Он включает в себя расчет, который суммирует результат 2 \*a и исходное значение, чтобы получить новое значение.

**Умножение на 3:**

* Это результат суммы (x×2⊕x), где x — исходный байт.
* Операция XOR происходит между результатом умножения на 2 и исходным байтом.

| Primul rând  9A:  2\*9A = 34 (calculat anterior).  00110100 XOR 10011010 = B0 | Al doilea rând  82:  2\*82 = 04 (calculat anterior).  3\*82 = 00000100 XOR 10000010 = 86 | Al treilea rând  72:  2\*72 = E4 (calculat anterior).  3\*72 = 11100100 XOR 01110010 = 95 | Al patrulea rând  67:  2\*67 = CE (calculat anterior).  3\*67 = 11001110 XOR 01100111 = A9 |
| --- | --- | --- | --- |
| E2:  2\*E2 = DF (calculat anterior).  3\*E2 =11000100 XOR DF = 3D | 63:  2\*63 = C6 (calculat anterior).  3\*63 = C6 XOR 63 = A5 | F2:  2\*F2 = E5(calculat anterior).  3\*F2 = E5 XOR F2 = 17 | A2:  2\*A2 = B7(calculat anterior).  3\*A2 = B7 XOR A2 = 15 |
| A0:  2\*A0 = 40(calculat anterior).  3\*A0 = 40 XOR A0 = E0 | 47:  2\*47 = 8E(calculat anterior).  3\*47 = 8E XOR 47 = C9 | 59:  2\*59 = B2 (calculat anterior).  3\*59 = B2 XOR 59 = EB | 58:  2\*58 = B0(calculat anterior).  3\*58 = B0 XOR 58 = E8 |
| 58:  2\*58 = B0(calculat anterior).  3\*58 = B0 XOR 58 = E8 | F0:  2\*F0 = EB (calculat anterior).  3\*F0 = EB XOR F0 = 1B | 77:  2\*77 = EE(calculat anterior).  3\*77 = EE XOR 77 = 99 | 20:  2\*20 = 40(calculat anterior).  3\*20 = 40 XOR 20 = 60 |

| B0 | 3D | E0 | E8 |
| --- | --- | --- | --- |
| 86 | A5 | C9 | 1B |
| 95 | 17 | EB | 99 |
| A9 | 15 | E8 | 60 |

Чтобы понять, как проводились эти вычисления, нам нужно посмотреть на каждый элемент b[i,j] в полученной таблице, где каждое значение b[i,j] получается сложением по модулю 256 (в шестнадцатеричном формате) соответствующих элементов в таблицах. 2\*а и 3\*а и текущий статус.

Эти значения обычно используются в специфичной для MixColumn матрице коэффициентов в AES, но здесь они представлены непосредственно как сумма членов промежуточной матрицы.

Calculam b:  
  
матрица после ShiftRow

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| 82 | 63 | 47 | f0 |
| 72 | f2 | 59 | 77 |
| 67 | A2 | 58 | 20 |

| 34 | DF | 40 | B0 |
| --- | --- | --- | --- |
| 04 | C6 | 8E | EB |
| E4 | E5 | B2 | EE |
| CE | B7 | B0 | 40 |

| B0 | 3D | E0 | E8 |
| --- | --- | --- | --- |
| 86 | A5 | C9 | 1B |
| 95 | 17 | EB | 99 |
| A9 | 15 | E8 | 60 |

 **b[0,0]**:

* Stare: 9A, 2\*a: 34, 3\*a: B0
* 9A+34+B0=382 mod 256=126
* **Rezultat:** 7E

 **b[1,0]**:

* Stare: 82, 2\*a: 04 , 3\*a: 86
* 82+04+86= 268mod256=12
* **Rezultat:** 0C

 **b[2,0]**:

* Stare: 72, 2\*a: E4, 3\*a: 95
* 72+E4+95= 491mod256=235
* **Rezultat:** EB

 **b[3,0]**:

* Stare: 67, 2\*a: CE, 3\*a: A9
* 67+CE+A9= 478mod256=222
* **Rezultat:** DE

 **b[0,1]**:

* Stare: e2, 2\*a: C6, 3\*a: 3d
* e2+DF+3d= 510mod256=254
* **Rezultat:** FE

 **b[1,1]**:

* Stare: 63, 2\*a: c6, 3\*a: A5
* 63+c6+a5= 462mod256=206
* **Rezultat:** CE

 **b[2,1]**:

* Stare: f2, 2\*a: E5, 3\*a: 17
* f2+E5+17 = 494mod256=238
* **Rezultat:** EE

 **b[3,1]**:

* Stare: a2, 2\*a: B7, 3\*a: 15
* a2+B7+15= 366mod256=110
* **Rezultat:** 6E

 **b[0,2]**:

* Stare: A0, 2\*a: 40, 3\*a: E0
* A0+40+40= 288mod256=32
* **Rezultat:** 20

 **b[1,2]**:

* Stare: 47, 2\*a: 8E, 3\*a: C9
* 47+8E+C9= 414mod256=158
* **Rezultat:** 9E

 **b[2,2]**:

* Stare: 59, 2\*a: B2, 3\*a: EB
* 59+B2+EB= 502mod256=246
* **Rezultat:** F6

 **b[3,2]**:

* Stare: 58, 2\*a: B0, 3\*a: E8
* 58+B0+E8= 496mod256=240
* **Rezultat:** F0

 **b[0,3]**:

* Stare: 58, 2\*a: B0, 3\*a: E8
* 58+B0+E8= 496mod256=240
* **Rezultat:** F0

 **b[1,3]**:

* Stare: f0, 2\*a: EB, 3\*a: 1B
* f0+EB+1B= 502mod256=246
* **Rezultat:** F6

 **b[2,3]**:

* Stare: 77, 2\*a: EE, 3\*a: 99
* 77+EE+99=510mod256=254
* **Rezultat:** FE

 **b[3,3]**:

* Stare: 20, 2\*a: 40, 3\*a: 60
* 20+40+60=192
* **Rezultat:** C0

Starea obtinuta:

| 7E | FE | 20 | F0 |
| --- | --- | --- | --- |
| 0C | CE | 9E | F6 |
| EB | EE | F6 | FE |
| DE | 6E | F0 | C0 |

**AddRoundKey**

Starea initiala:

| 7E | FE | 20 | F0 |
| --- | --- | --- | --- |
| 0C | CE | 9E | F6 |
| EB | EE | F6 | FE |
| DE | 6E | F0 | C0 |

Cheia utilizata: (XOR cu ultimile 4 rinduri din cheie):

| b2 | 4b | b6 | 2a |
| --- | --- | --- | --- |
| D6 | 2E | C4 | 59 |
| A2 | 41 | B6 | 34 |
| 92 | 70 | 84 | 07 |

Starea obtinuta:

| СС | B5 | 96 | DA |
| --- | --- | --- | --- |
| DA | E0 | 5A | AF |
| 49 | AF | 40 | CA |
| 4C | 1E | 74 | C7 |

7E 01111110

XOR

B2 10110010

--------

CC 11001100 (в двоичном)

Зашифрованное сообщение: ccb596dadae05aaf49af40ca4c1e74c7 => ASCII => ÌµÚÚàZ¯I¯@ÊLtÇ

**Дешифрование:**

Обратный процесс можно использовать с помощью приложений AddRoundKey, InvMixColumn, InvShiftRow и InvByteSub для получения исходного сообщения.

Запись сообщения: ccb596dadae05aaf49af40ca4c1e74c7

Ключ (128 бит)

| 74 | 68 | 75 | 6E | 64 | 65 | 72 | 73 | 74 | 6F | 72 | 6D | 30 | 31 | 32 | 33 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

W:

| ***0*** | 74 | 68 | 75 | 6E |
| --- | --- | --- | --- | --- |
| ***1*** | 64 | 65 | 72 | 73 |
| ***2*** | 74 | 6F | 72 | 6D |
| ***3*** | 30 | 31 | 32 | 33 |
| ***4*** | b2 | 4b | b6 | 2a |
| ***5*** | D6 | 2E | C4 | 59 |
| ***6*** | A2 | 41 | B6 | 34 |
| ***7*** | 92 | 70 | 84 | 07 |

**Runda 1:**

**AddRoundKey**

Starea initiala:

| СС | B5 | 96 | DA |
| --- | --- | --- | --- |
| DA | E0 | 5A | AF |
| 49 | AF | 40 | CA |
| 4C | 1E | 74 | C7 |

Cheia utilizata: XOR

| b2 | 4b | b6 | 2a |
| --- | --- | --- | --- |
| D6 | 2E | C4 | 59 |
| A2 | 41 | B6 | 34 |
| 92 | 70 | 84 | 07 |

Starea obtinuta:

| 7E | FE | 20 | F0 |
| --- | --- | --- | --- |
| 0C | CE | 9E | F6 |
| EB | EE | F6 | FE |
| DE | 6E | F0 | C0 |

**InvMixColumn**

Чтобы получить последний старый 9A E2 A0 58 82 63 47 F0 72 F2 59 77 67 A2 58 20 в вашем InvMixColumn, вы вычислите элемент, соответствующий формуле pentru inversarea coloanelor в AES.

Starea obtinuta:

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| 82 | 63 | 47 | f0 |
| 72 | f2 | 59 | 77 |
| 67 | A2 | 58 | 20 |

**InvShiftRow**

InvShiftRow используется в процессе расшифровки алгоритма AES и позволяет найти обратное преобразование ShiftRow в шифровании.

Во время текущей работы ваш ранд-старт разваливается, когда число позиций различается, в функции рандула нужно включить:

Ряд 0 немодифицирован.

Ряд 1 должен быть заменен на место, где вы находитесь.

Ряд 2 может быть заменен на две позиции, где вы находитесь.

Ряд 3 будет заменен на три позиции, где вы находитесь.

Starea initiala:

| 9A | e2 | A0 | 58 | >>>0 |
| --- | --- | --- | --- | --- |
| 82 | 63 | 47 | f0 | >>>1 |
| 72 | f2 | 59 | 77 | >>>2 |
| 67 | A2 | 58 | 20 | >>>3 |

Цель InvShiftRow — отменить эффект диффузии, созданный ShiftRow во время шифрования, восстановить исходный порядок элементов и подготовить состояние для последующих операций дешифрования.

Starea obtinuta:

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| 63 | 47 | f0 | 82 |
| 59 | 77 | 72 | f2 |
| A2 | 72 | f2 | 67 |

**InvByteSub:**

InvByteSub — это операция обратной подстановки, используемая в процессе расшифровки AES и предназначенная для отмены эффекта операции ByteSub, примененной во время шифрования. На этом этапе каждый байт состояния заменяется с использованием таблицы обратной замены, известной как InvS-Box.

Starea initiala:

| 9A | e2 | A0 | 58 |
| --- | --- | --- | --- |
| 63 | 47 | f0 | 82 |
| 59 | 77 | 72 | f2 |
| A2 | 72 | f2 | 67 |

Starea obtinuta:

| 37 | 3B | 47 | 5E |
| --- | --- | --- | --- |
| 17 | 11 | 00 | 16 |
| 15 | 02 | 1E | 04 |
| 1A | 5E | 54 | 0A |

**AddRoundKey (utimul)**

В процессе расшифровки AES операция AddRoundKey аналогична операции шифрования и играет решающую роль. Его цель — объединить текущее состояние с соответствующим ключом раунда, но при расшифровке этот шаг применяется в порядке, обратном шифрованию, начиная с последнего раунда и продвигаясь к первому.

Starea initiala:

| 37 | 3B | 47 | 5E |
| --- | --- | --- | --- |
| 17 | 11 | 00 | 16 |
| 15 | 02 | 1E | 04 |
| 1A | 5E | 54 | 0A |

Cheia utilizata:

| 74 | 68 | 75 | 6E |
| --- | --- | --- | --- |
| 64 | 65 | 72 | 73 |
| 74 | 6F | 72 | 6D |
| 30 | 31 | 32 | 33 |

Starea obtiunta:

| 43 | 53 | 32 | 30 |
| --- | --- | --- | --- |
| 73 | 74 | 72 | 65 |
| 61 | 6D | 6C | 69 |
| 6E | 65 | 64 | 39 |

Mesajul decriptat: 43 53 32 30 73 74 72 65 61 6D 6C 69 6E 65 64 39 (CS20streamlined9)

Вывод: окончательный результат — это результат успешно зашифрованный, CS20streamlined9, идентичный исходному результату. Это подтверждение выполненного алгоритма AES и ключ функций сработал правильно, шифруя и дешифруя сообшение в подходящем виде.

**Sarcina B) Mercle Signature, One time Lamport Signature**

**Подпись Лампорта (Lamport Signature)** – это одноразовая схема цифровой подписи, основанная на использовании криптографически стойких хеш-функций. Она устойчива к квантовым атакам и применима для защиты критически важных данных. Из-за своей природы одноразовой подписи её удобно использовать в комбинации с деревом Меркла, что позволяет эффективно управлять множеством подписей.

#### **Основные элементы:**

* **Документ:** данные, которые необходимо подписать.
* **Секретный ключ:** генерируется для каждой подписи и используется только один раз.
* **Публичный ключ:** производный от секретного ключа, открыто доступный для проверки подписи.

#### **Формирование подписи Лампорта:**

1. **Секретный ключ:** создаётся массив случайных пар чисел, где количество пар соответствует длине документа в двоичном виде (или длине его хеша).  
   Например, для документа длиной 256 битов будет создано 256 пар (ai,bi).
2. **Публичный ключ:** это хеши всех элементов из секретного ключа:  
   Yi=H(ai),H(bi).
3. **Создание подписи:**
   * Документ или его хеш переводится в двоичный формат.
   * Каждому биту документа соответствует одна пара из секретного ключа:
     + Если бит равен 0, выбирается ai​.
     + Если бит равен 1, выбирается bi​.
   * Результат — массив, содержащий выбранные значения, служит подписью.
4. **Проверка подписи:**
   * Проверяющая сторона использует публичный ключ и документ для проверки.
   * Она вычисляет хеши элементов подписи и сравнивает их с публичным ключом.
   * Если значения совпадают, подпись считается действительной.

**Пример.**

Пусть p=3931 — простое число, а 3 принадлежит полю  *Zp=*3931 — примитивному элементу. Определим f(x) = 3*x mod* 3931*:*

Если Пользователь\_B хочет подписать трехбитное сообщение, он выбирает (тайно) шесть случайных чисел:

*y*1*,*0 = 5741 *y*2*,*0 = 751 *y*3*,*0 = 991

*y*1*,*1 = 9803 *y*2*,*1 = 467*y*3*,*1 = 62254

Затем вычислите их изображение по функции f:

*z*1*,*0 *= 3^y*1*,*0  *mod* 3931 =  *3^5741* *mod* 3931 = 1304 *z*1*,*0 = 1304 *z*2,0 = 1966 *z*3*,*0 = 2887

*z*1,1 = 1778 *z*2*,*1 = 3607 *z*3*,*1 = 3440

Эти номера z являются общедоступными.

Предположим, Пользователь\_B хочет подписать сообщение x = (0, 1, 0). Его подпись

(*y*1*0, y*2*,*1*, y*3*,*0) = (5741, 467, 991)

Для проверки подписи достаточно установить, что:

35741 mod 3931 = 1304; 3467 mod 3931 = 3607; 3991 mod 3931 = 2887

все вычисления выполняются по модулю 3931 .

Подпись невозможно имитировать, поскольку f не имеет обратной. Протокол подписи можно использовать только для одного сообщения.

#### 

#### **Ограничение подписи Лампорта**

Она является **одноразовой** — после использования секретного ключа его нельзя применять повторно, так как это открывает путь для подделки подписи.

Чтобы устранить этот недостаток, применяется **дерево Меркла**:  
множество одноразовых подписей объединяется в иерархическую структуру, где корневой хеш служит общим публичным ключом.

### **Дерево Меркла (Merkle Tree)**

**Дерево Меркла** – это структура данных, представляющая собой бинарное дерево, где каждый узел является хэшем. Оно позволяет эффективно проверять подлинность отдельных данных без необходимости хранить весь их массив.

#### **Основные элементы:**

* **Листовые узлы:** хеши данных или их подписей.
* **Внутренние узлы:** хеши, полученные из комбинации двух дочерних узлов.
* **Корневой узел (root hash):** финальный хеш, объединяющий все данные дерева.

#### **Построение дерева:**

1. **Инициализация:** для каждого блока данных вычисляется хеш, который становится листом дерева.
2. **Создание уровней:** хеши листьев объединяются попарно, и для каждой пары вычисляется хеш, формирующий следующий уровень.
3. **Корневой хеш:** процесс повторяется до тех пор, пока не останется единственный хеш — корневой узел дерева.

Если на каком-то уровне остаётся нечетное число узлов, последний узел дублируется. Это необходимо, чтобы корректно завершить построение дерева.

#### **Проверка данных с помощью дерева Меркла:**

Для проверки подлинности блока данных используется **Merkle Proof** — набор хешей, необходимых для восстановления пути от листа до корневого узла.

**Проверка выполняется следующим образом:**

1. Пользователь хеширует документ.
2. Полученный хеш комбинируется с хешами из Merkle Proof на каждом уровне дерева.
3. Если итоговый хеш совпадает с корневым узлом, документ считается подлинным.

### **Преимущества дерева Меркла:**

1. **Эффективная проверка:** позволяет проверять подлинность данных, загружая только часть дерева.
2. **Экономия пространства:** достаточно хранить лишь корневой хеш.
3. **Безопасность:** малейшие изменения в данных приводят к изменению корневого хеша.

### **Недостатки:**

1. **Одноразовое построение:** при изменении данных дерево нужно создавать заново.
2. **Сложность обновления:** внесение изменений требует перестройки всех связанных узлов.  
     
   **Пример генерации подписи Меркла**

Пример пошагового вычисления подписи Меркла для 8 сообщений.  
  
**Дано:**

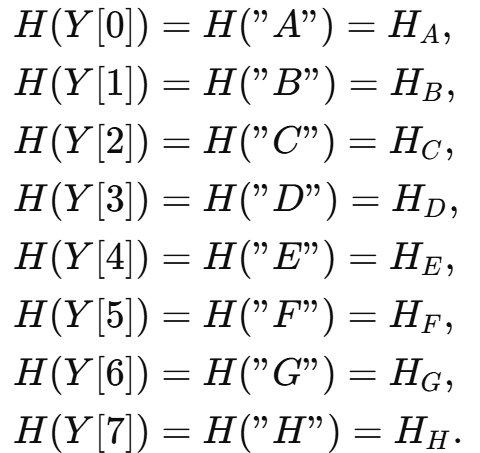
Сообщения:

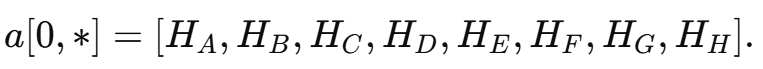
* Y[0]="A"
* Y[1]="B"
* Y[2]="C"
* Y[3]="D"
* Y[4]="E"
* Y[5]="F"
* Y[6]="G"
* Y[7]="H"

Хэш-функция: H(x) = "H" + x.

**Шаг 1: Вычисление хэшей для листьев (уровень a[0,∗])**

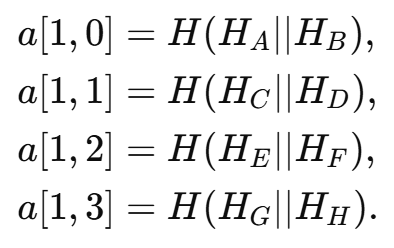
Для каждого сообщения вычисляем хэш:



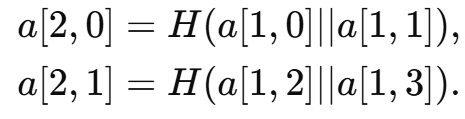
Листья дерева:  
.

### **Шаг 2: Вычисление хэшей на следующем уровне (уровень a[1,∗])**

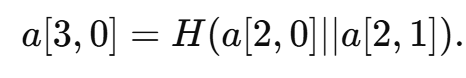
Попарно объединяем и хешируем листья:

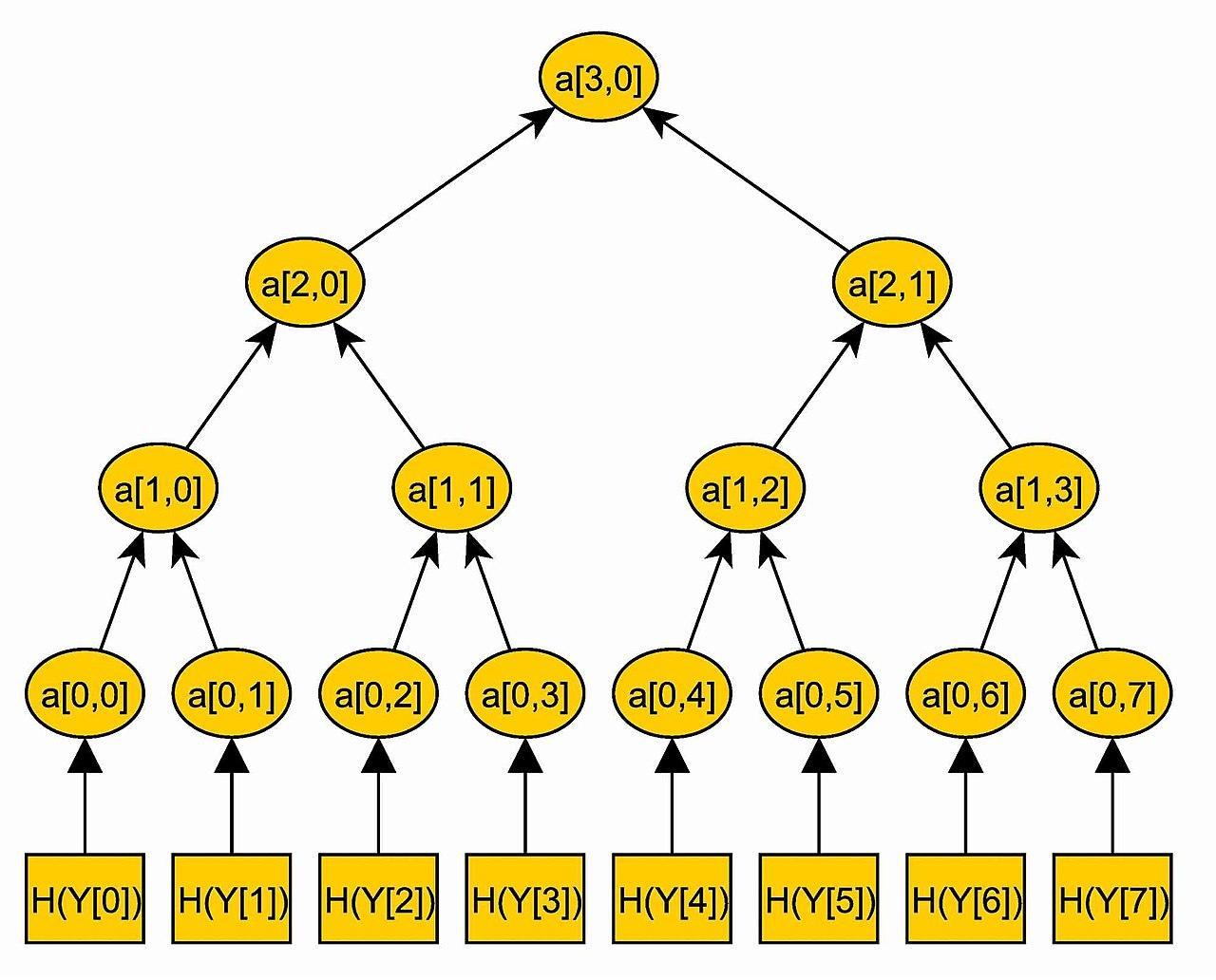
  
  
**Шаг 3: Вычисление хэшей на уровне a[2,∗]**

Объединяем узлы с предыдущего уровня:

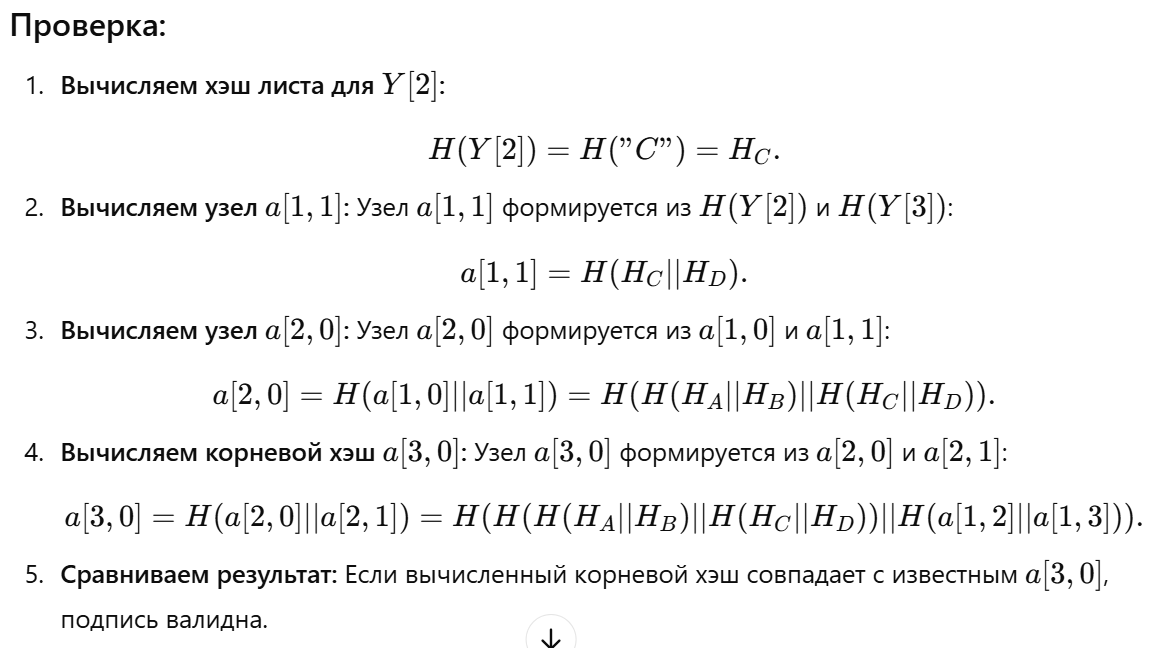
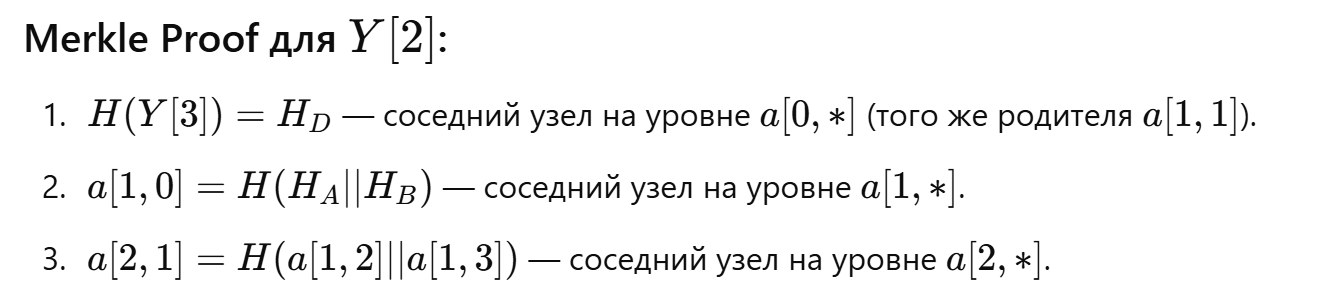
  
**Шаг 4: Вычисление корневого хэша (уровень a[3,0])**

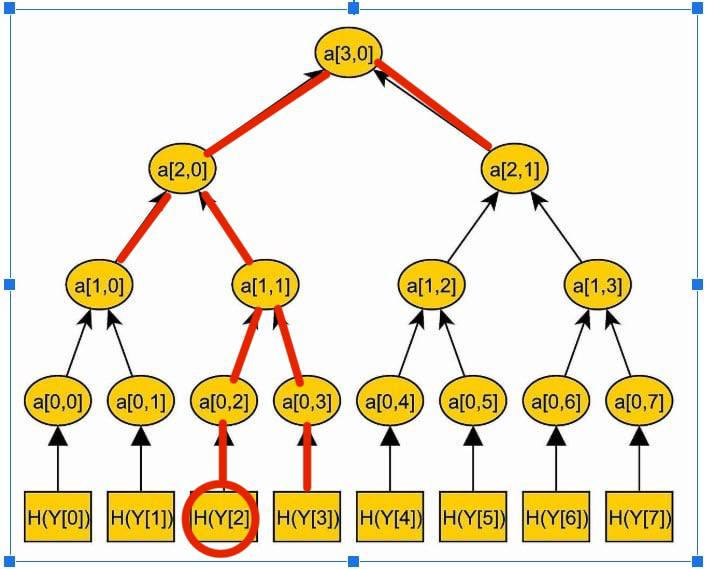
Объединяем два оставшихся узла:



  
давайте проведем этап проверки для сообщения Y[2]. Это сообщение находится в третьем листе (a[0,2]).  
  
**Дано:**

Сообщение: Y[2]="C".  
Корневой хэш: a[3,0], полученный в процессе построения дерева.

Для проверки нам нужно предоставить **доказательство Меркле** (Merkle Proof), содержащее соседние узлы, необходимые для вычисления корневого хэша.  
  


  
В дереве Меркла соседние ветки берутся для проверки целостности и подлинности данных. Когда вы хотите проверить конкретный элемент (например, H(Y[2])), вам нужно получить путь к корню дерева. Этот путь включает не только сам целевой элемент, но и хэши соседних узлов (называемые **доказательствами включения**, чтобы обеспечить целостность данных).  
  
Так как при изменении любого документа, root-hash изменится, то проверка таким образом гарантирует, что ни 1 документ не был изменён после отправки.

### **Заключение**

Подпись Лампорта в комбинации с деревом Меркла позволяет эффективно и безопасно управлять множеством одноразовых подписей, предоставляя единый корневой хэш для проверки их подлинности. Эта схема широко используется в блокчейне, системах контроля версий и цифровых подписях, где важны безопасность, экономия ресурсов и удобство проверки.