# MÃ HÓA HIỆN ĐẠI – MÃ HÓA ĐỐI XỨNG P.2

## 9. Thuật toán AES - Advanced Encryption Standard

- Còn gọi là thuật toán Rijndael, là một **mật mã khối** lặp đi **lặp lại với độ dài khối thay đổi** và **độ dài khóa khác nhau.**
- Chiều dài của khóa và chiều dài của khối có thể độc lập hoặc trùng nhau, thuộc vào 1 trong 3 giá trị: **128, 192 hoặc 256 bit.**
- Trong AES, tất cả các hoạt động được thực hiện trên các byte 8-bit. Các phép toán số học gồm phép cộng, nhân và chia được thực hiện trong trường hữu hạn GF($2^8$).
- Các phép biến đổi khác nhau hoạt động với một kết quả trung gian được gọi là *Trạng thái (State).* Trạng thái có thể được biểu diễn dưới dạng một mảng byte hình chữ nhật.
- Để đơn giản hóa và chuẩn hóa quá trình xử lý, AES quy định ma trận trạng thái luôn có **4 dòng**, bất kể kích thước khối & kích thước khóa là bao nhiêu. Số cột sẽ thay đổi tùy theo kích thước khối:
  - **Đối với khối bản rõ:**
    - Được biểu diễn dưới dạng một ma trận A, với các phần tử A(i,j), trong đó i là chỉ số dòng (từ 0 đến 3), j là chỉ số cột (từ 0 đến $N_{b-1}$).
    - **số cột được ký hiệu là $N_b$** và bằng chiều dài của khối chia cho 32 *(khối 128, 192, và 256 bit). (Vì có 4 dòng, mỗi ô chiếm 8 bit = 1 byte → Số bit của 1 cột là 8x4 = 32 bit, để tìm số cột lấy tổng số bit chia cho số bit của 1 cột)*
      - Khối 128 bit → $N_b = 4$
      - Khối 192 bit → $N_b = 6$
      - Khối 256 bit → $N_b = 8$
  - **Đối với Khóa mã hóa:**
    - Được biểu diễn dưới dạng một ma trận K, với các phần tử K(i,j), trong đó i là chỉ số dòng (từ 0 đến 3), j là chỉ số cột (từ 0 đến $N_{k-1}$).
    - **Số cột được ký hiệu là $N_k$** và bằng chiều dài của khóa chia cho 32 (khóa 128, 192, và 256 bit).
      - Khóa 128 bit → $N_k = 4$
      - Khóa 192 bit → $N_k = 6$
      - Khóa 256 bit → $N_k = 8$

| | | | | | |
|---|---|---|---|---|---|
| A(0,0) | A(0,1) | A(0,2) | A(0,3) | A(0,4) | A(0,5) |
| A(1,0) | A(1,1) | A(1,2) | A(1,3) | A(1,4) | A(1,5) |
| A(2,0) | A(2,1) | A(2,2) | A(2,3) | A(2,4) | A(2,5) |
| A(3,0) | A(3,1) | A(3,2) | A(3,3) | A(3,4) | A(3,5) |

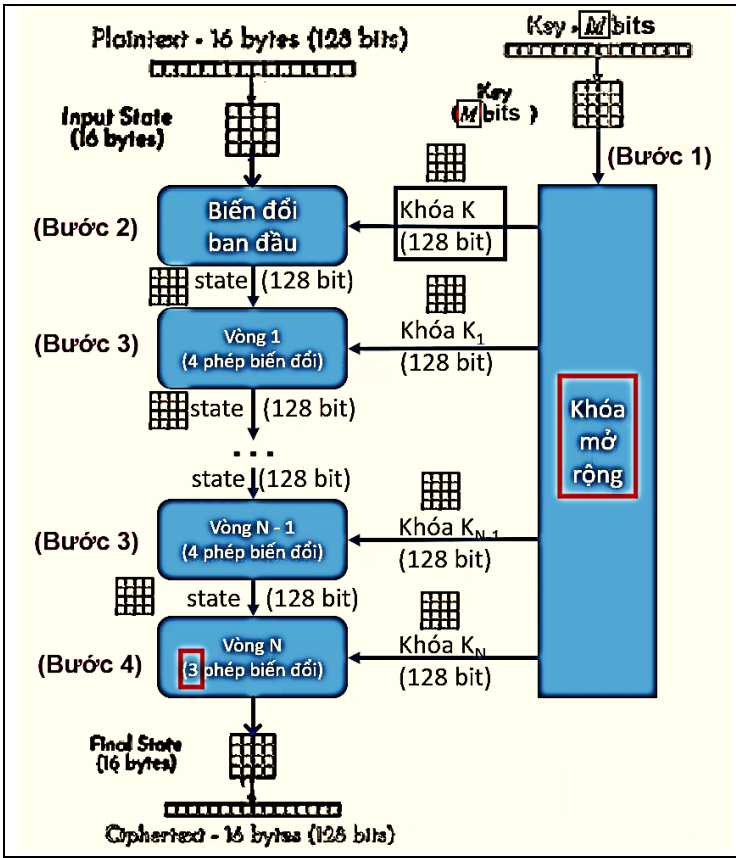
| | | | |
|---|---|---|---|
| K(0,0) | K(0,1) | K(0,2) | K(0,3) |
| K(1,0) | K(1,1) | K(1,2) | K(1,3) |
| K(2,0) | K(2,1) | K(2,2) | K(2,3) |
| K(3,0) | K(3,1) | K(3,2) | K(3,3) |

**Ví dụ về biểu diễn $N_b = 6$ và $N_k = 4$**

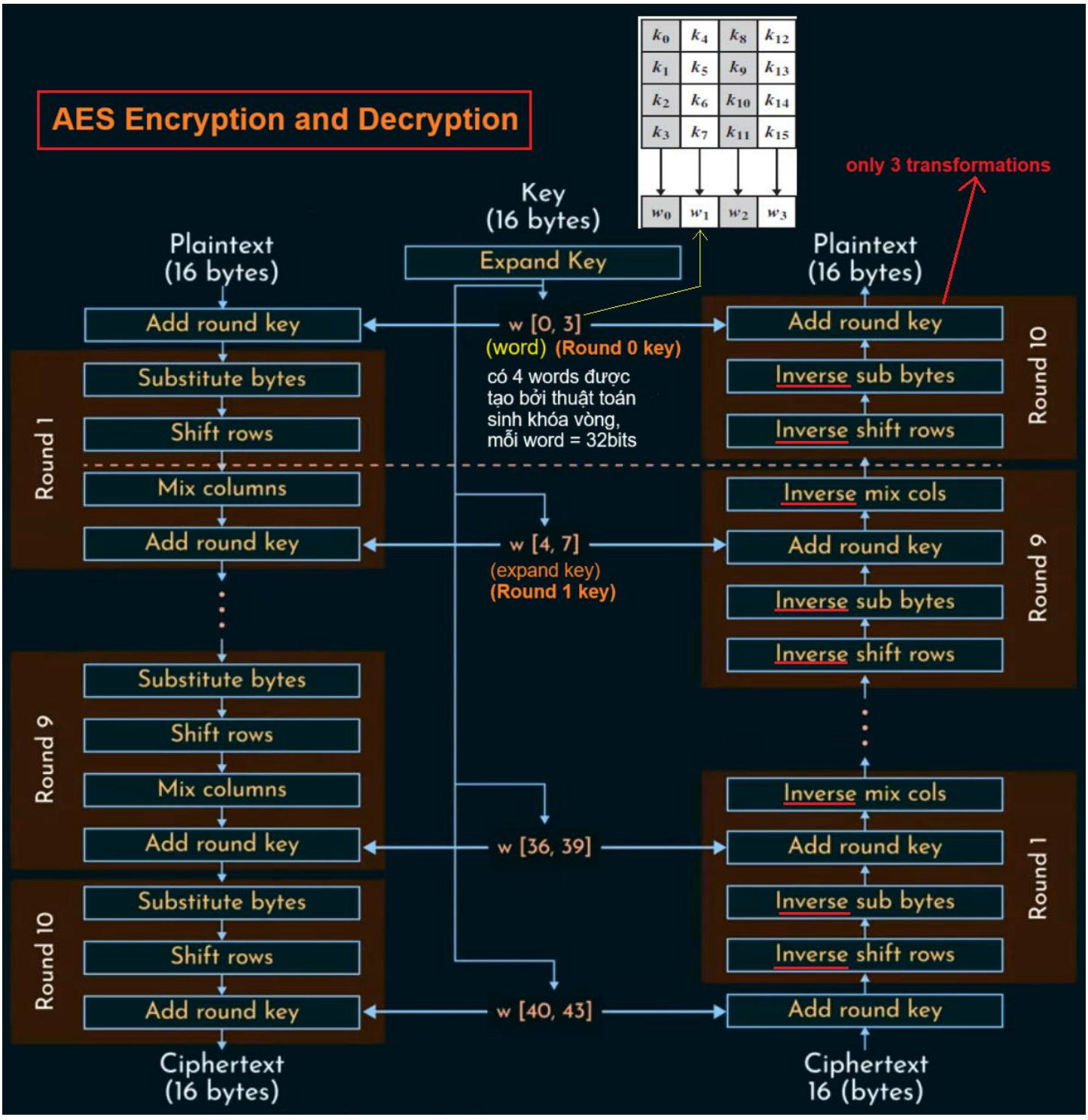
- **Số chu kỳ $N_r$ (số vòng – rounds)** của Rijdael là một hàm của chiều dài khối và khoá

| Số cột khối / Số cột khóa | $N_b = 4$ (128 bit) | $N_b = 6$ (192 bit) | $N_b = 8$ (256 bit) |
|---|---|---|---|
| **$N_k = 4$ (128 bit)** | $N_r = 10$ | $N_r = 12$ | $N_r = 14$ |
| **$N_k = 6$ (192 bit)** | $N_r = 12$ | $N_r = 12$ | $N_r = 14$ |
| **$N_k = 8$ (256 bit)** | $N_r = 14$ | $N_r = 14$ | $N_r = 14$ |

- **Vì AES chỉ mã hóa các khối có kích thước cố định** (128 bit, 192 bit, 256 bit), nên ta cần **kết hợp với các chế độ hoạt động** của mã hóa khối (mode), gồm **CBC** (Cipher Block Chaining), **CTR** (counter mode).
- **Quá trình mã hoá:** Giải thuật AES thực hiện mã hóa khối dữ liệu bản rõ, gồm các bước xử lý chính như sau:




- **Bước 1: Mở rộng khóa** - thực hiện việc **sinh các khóa vòng** (Round key) từ khóa chính AES – **có kích thước M** (bits), sử dụng thủ tục sinh khóa Rijndael, được **dùng trong các vòng lặp, các khóa vòng này sẽ phải có kích thước bằng với kích thước của khối**.
  - Trong tiêu chuẩn AES, kích thước khóa ban đầu **ít nhất bằng kích thước khối**, thường là lớn hơn.
  - Số lượng khóa vòng: $\mathbf{N_r + 1}$
  - lý do cộng 1 là vì ta cần 1 khóa cho vòng khởi tạo, $N_r$ khóa cho các vòng lặp.
- **Bước 2**: Vòng khởi tạo (round 0) thực hiện hàm AddRoundKey, trong đó **mỗi byte trong state được thực hiện phép XOR với khóa vòng.**
- **Bước 3**: Các vòng lặp chính, trong đó mỗi vòng thực hiện 4 hàm biến đổi dữ liệu như sau:
  - **SubBytes** là hàm thay thế phi tuyến tính, trong đó mỗi byte trong **state** được thay thế bằng một byte khác sử dụng **bảng tham chiếu S-box;**
  - **ShiftRows** là hàm dịch dòng, trong đó mỗi dòng trong **state** được dịch một số bước theo chu kỳ;
  - **MixColumns** là làm trộn các cột trong state, **kết hợp 4 bytes trong mỗi cột.**
  - **AddRoundKey** là hàm kết hợp state với khóa vòng sử dụng phép XOR.
- **Bước 4**: Vòng lặp cuối tương tự các vòng lặp chính, nhưng chỉ thực hiện 3 hàm biến đổi dữ liệu, bao gồm **SubBytes, ShiftRows và AddRoundKey**


AES Encryption and Decryption
$k_0$ $k_4$ $k_8$ $k_{12}$
$k_1$ $k_5$ $k_9$ $k_{13}$
$k_2$ $k_6$ $k_{10}$ $k_{14}$
$k_3$ $k_7$ $k_{11}$ $k_{15}$
$w_0$ $w_1$ $w_2$ $w_3$
only 3 transformations
Key
(16 bytes)
Plaintext
(16 bytes)
Expand Key
Add round key
w [0, 3]
(word) (Round 0 key)
có 4 words được
tạo bởi thuật toán
sinh khóa vòng,
mỗi word = 32bits
Substitute bytes
Shift rows
Mix columns
Add round key
Round 1
w [4, 7]
(expand key)
(Round 1 key)
Substitute bytes
Shift rows
Mix columns
Add round key
Round 9
w [36, 39]
Substitute bytes
Shift rows
Add round key
Round 10
w [40, 43]
Ciphertext
(16 bytes)
Plaintext
(16 bytes)
Add round key
Inverse sub bytes
Inverse shift rows
Round 10
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 9
Inverse mix cols
Add round key
Inverse sub bytes
Inverse shift rows
Round 1
Add round key
Ciphertext
16 (bytes)

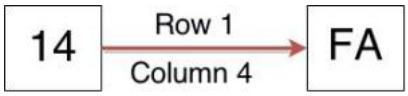
## a/ Phép biến đổi ByteSub (subtitute bytes – thay thế bytes)

- là một phép thay thế byte phi tuyến tính được thực hiện cho từng byte trạng thái một cách độc lập.
- Phép SubBytes **thay thế mỗi byte trong state bằng 1 byte trong bảng S-box** bên dưới.

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

**Ví dụ:**

- Byte {95} được thay thế thành {2A} (giá trị tại hàng 9, cột 5 của bảng S-box) → SubBytes({95}) = {2A}
- Tương tự, SubByte({59}) = {CB}




- **Bản chất,** phép sub bytes gồm 2 bước:
  - **B1: Nghịch đảo trong trường hữu hạn GF($2^8$)**
    - Mỗi byte trong trạng thái (state) được xem như một phần tử trong trường Galois GF($2^8$).
    - Byte đó được tính **nghịch đảo** trong GF($2^8$) theo modulo đa thức bất khả quy m(x) = $x^8 + x^4 + x^3 + x + 1$ (tìm nghịch đảo của từng byte trạng thái theo modulo $x^8 + x^4 + x^3 + x + 1$).
    - Nếu byte đầu vào là 00, nghịch đảo của nó được định nghĩa là 00 (để tránh trường hợp không xác định).
  - **B2: Biến đổi Affine trong GF(2):**
    - Sau khi có nghịch đảo, áp dụng phép biến đổi affine như sau:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

  - Trong đó:
    - $[x_0, x_1, ..., x_7]$ là biểu diễn nhị phân của byte nghịch đảo (từ bước 1).
    - Ma trận 8x8 là ma trận cố định dùng cho phép biến đổi Affine.
    - Vector [1,1,0,0,0,1,1,0] được cộng vào kết quả.
    - Các phép toán (nhân ma trận và cộng) được thực hiện trong GF(2), tức là:
      - Cộng: Phép XOR (0 + 0 = 0, **1 + 1 = 0**, 0 + 1 = 1).
      - Nhân: Phép AND (0 × 0 = 0, 1 × 1 = 1, 0 × 1 = 0).
    - Kết quả thu được là $[y_0, y_1, ..., y_7]$.
- **Tuy nhiên, để tiết kiệm thời gian, người ta sử dụng bảng S-box để tra cứu (như giải thích ở trên), thay vì tính toán.**
- Đảo ngược ByteSub (inverse sub byte) là ứng dụng thay thế byte theo **bảng nghịch đảo (bên dưới).** Điều này thu được bằng cách đảo ngược ánh xạ affine và đảo ngược phép nhân trong GF ($2^8$).

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

- **Phép biến đổi affine nghịch đảo được biểu diễn như sau:**

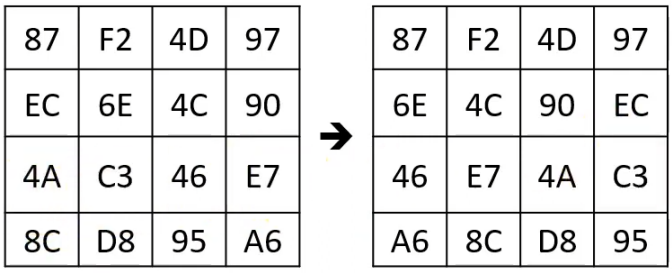
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix}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

## b/ Phép biến đổi ShiftRow

- các hàng trạng thái được chuyển theo chu kỳ sang các giá trị khác nhau.
- Thực hiện phép dịch trái đối với các bytes ở mỗi hàng, theo quy tắc như sau:
  - Dòng số 0 không di chuyển. Dòng 1 được dịch chuyển bởi $C_1$ byte, dòng 2 bởi $C_2$ byte, dòng 3 bởi $C_3$ byte. Giá trị của $C_1$, $C_2$ và $C_3$ phụ thuộc vào $N_b$.

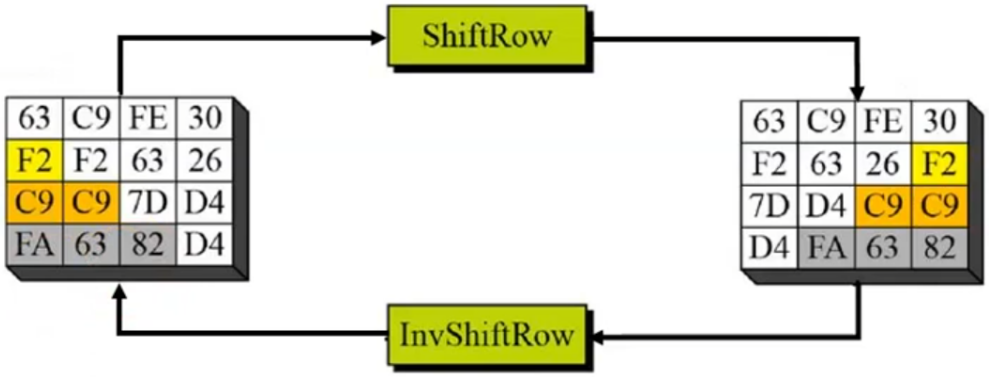
| $N_b$ | $C_1$ | $C_2$ | $C_3$ |
|---|---|---|---|
| 4 | 1 | 2 | 3 |
| 6 | 1 | 2 | 3 |
| 8 | 1 | 3 | 4 |

- **Ví dụ:**

| | |
|---|---|
| - Hàng đầu không thay đổi.<br>- Hàng hai, dịch vòng **trái** 1-byte.<br>- Hàng ba, dịch vòng **trái** 2 byte.<br>- Hàng tư, dịch vòng **trái** 3-byte. |  |



- **Đảo ngược đối với ShiftRow (Inverse ShiftRow)**: **dịch phải** theo chu kỳ của ba hàng dưới lần lượt $N_b - C_1$, $N_b - C_2$ và $N_b - C_3$ byte, sao cho byte ở vị trí thứ *j* trong hàng *i* di chuyển đến vị trí $(j + N_b - C_i) \bmod N_b$. Hàng đầu tiên vẫn giữ nguyên.
- **Ví dụ:**
  - **Chẳng hạn với $N_b = 4$:**
    - **Hàng 0:** Dịch 0 bước.
    - **Hàng 1:** Dịch phải $N_b - C_1 = 4 - 1 = 3$ bước.
    - **Hàng 2:** Dịch phải $N_b - C_2 = 4 - 2 = 2$ bước.
    - **Hàng 3:** Dịch phải $N_b - C_3 = 4 - 3 = 1$ bước.




## c/ Phép biến đổi MixColumn

- Được định nghĩa bằng **phép nhân ma trận** sau

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

- $[a_0, a_1, a_2, a_3]$ là một cột của trạng thái (state) trước MixColumns.
- $[b_0, b_1, b_2, b_3]$ là cột sau khi biến đổi.

| | | | |
|---|---|---|---|
| 2 | 3 | 1 | 1 |
| 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 3 |
| 3 | 1 | 1 | 2 |

*

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| 6E | 4C | 90 | EC |
| 46 | E7 | 4A | C3 |
| A6 | 8C | D8 | 95 |

=

| | | | |
|---|---|---|---|
| ? | | | |
| | | | |
| | | | |
| | | | |

$\{02\} * \{87\} \oplus \{03\} * \{6E\} \oplus \{01\} * \{46\} \oplus \{01\} * \{A6\}$

$02 = 0000\ 0010 = X$

$87 = 1000\ 0111 = X^7 + X^2 + X + 1$

$02 * 87 = X * (X^7 + X^2 + X + 1)$

$= X^8 + X^3 + X^2 + X$

$= X^4 + \cancel{X^3} + \cancel{X} + 1 + \cancel{X^3} + X^2 + \cancel{X}$

$= X^4 + X^2 + 1$

$= 0001\ 0101$

$03 = 0000\ 0011 = X + 1$

$6E = 0110\ 1110 = X^6 + X^5 + X^3 + X^2 + X$

$03 * 6E = (X+1) * (X^6 + X^5 + X^3 + X^2 + X)$

$= X^7 + \cancel{X^6} + X^4 + \cancel{X^3} + \cancel{X^2} + \cancel{X^6} + X^5 + \cancel{X^3} + \cancel{X^2} + X$

$= X^7 + X^5 + X^4 + X$

$= 1011\ 0010$

$01 * 46 = 46 = 0100\ 0110$

$01 * A6 = A6 = 1010\ 0110$

02 * 87 = 0 0 0 1  0 1 0 1
03 * 6E = 1 0 1 1  0 0 1 0
01 * 46 = 0 1 0 0  0 1 1 0
01 * A6 = 1 0 1 0  0 1 1 0

= 0 1 0 0  0 1 1 1 = {47}
4  7

| | | | |
|---|---|---|---|
| 2 | 3 | 1 | 1 |
| 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 3 |
| 3 | 1 | 1 | 2 |

*

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| 6E | 4C | 90 | EC |
| 46 | E7 | 4A | C3 |
| A6 | 8C | D8 | 95 |

=

| | | | |
|---|---|---|---|
| 47 | | | |
| ? | | | |
| | | | |
| | | | |

{01} * {87} ⊕ {02} * {6E} ⊕ {03} * {46} ⊕ {01} * {A6}

01 * 87 = 87 = 1000 0111

02 = 0000 0010 = X
6E = 0110 1110 = $X^6 + X^5 + X^3 + X^2 + X$
02 * 6E = X * ($X^6 + X^5 + X^3 + X^2 + X$)
= $X^7 + X^6 + X^4 + X^3 + X^2$
= 1101 1100

03 = 0000 0110 = X + 1
46 = 0100 0110 = $X^6 + X^2 + X$
03 * 46 = (X + 1) * ($X^6 + X^2 + X$)
= $X^7 + X^3 + X^2 + X^6 + X^2 + X$
= $X^7 + X^6 + X^3 + X$
= 1100 1010

01 * A6 = A6 = 1010 0110

01 * 87 = 1 0 0 0  0 1 1 1
02 * 6E = 1 1 0 1  1 1 0 0
03 * 46 = 1 1 0 0  1 0 1 0
01 * A6 = 1 0 1 0  0 1 1 0

= 0 0 1 1  0 1 1 1 = {37}
3  7

- Tương tự với các ô còn lại của ma trận. Cuối cùng ta được ma trận trạng thái sau:

| | | | |
|---|---|---|---|
| 2 | 3 | 1 | 1 |
| 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 3 |
| 3 | 1 | 1 | 2 |

*

| | | | |
|---|---|---|---|
| 87 | F2 | 4D | 97 |
| 6E | 4C | 90 | EC |
| 46 | E7 | 4A | C3 |
| A6 | 8C | D8 | 95 |

=

| | | | |
|---|---|---|---|
| 47 | 40 | A3 | 4C |
| 37 | D4 | 70 | 9F |
| 94 | E4 | 3A | 42 |
| ED | A5 | A6 | BC |

- **Đảo ngược MixColumn (Inverse Mix Column)** tương tự như MixColumn, chỉ khác là nhân với một ma trận khác.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix}$$

### d/ Phép AddRoundKey

- Được thực hiện **XOR của khoá vòng tương ứng với trạng thái hiện tại.**
- Chiều dài của khoá vòng bằng chiều dài của khối $N_b$.
- **AddRoundKey là tự nghịch đảo:** vì phép XOR có tính chất tự nghịch đảo, nghĩa là nếu bạn áp dụng XOR hai lần với cùng một giá trị, bạn sẽ quay lại giá trị ban đầu
    - Mã hóa: State⊕Round Key
    - Giải mã: (State⊕Round Key)⊕Round Key=State
- Vì vậy, **Inverse AddRoundKey** chính là AddRoundKey

### e/ Khóa mở rộng (Key Expansion) & Sinh khóa vòng

- **Khóa mã hóa (Cipher Key/khoá đầu vào) có kích thước $M = 4 \times N_k$ byte**, trong đó **$N_k$ là số từ (word) của khóa (còn gọi là số cột của khóa):**
    - $N_k = 4$ (khóa 128 bit).
    - $N_k = 6$ (khóa 192 bit).
    - $N_k = 8$ (khóa 256 bit).
- **Khóa mở rộng** là một mảng tuyến tính gồm **$N_b \times (N_r + 1)$ từ (word),** trong đó:
    - $N_b$: Số cột của trạng thái (thường là 4 trong AES-128).
    - $N_r$: Số vòng (10, 12, hoặc 14).
    - Với AES-128 ($N_b = 4$, $N_r = 10$): Số từ trong Khóa mở rộng = $N_b \times (N_r + 1) = 4 \times (10 + 1) = 44$ từ.
- **Lựa chọn khóa vòng:** Các khóa vòng được **lấy từ Khóa mở rộng**:
    - **Khóa vòng đầu tiên:** Lấy $N_b$ từ đầu tiên (từ W[0] đến W[$N_b - 1$]).
    - **Khóa vòng thứ hai:** Lấy $N_b$ từ tiếp theo (từ W[$N_b$] đến W[$2N_b - 1$]), v.v.
    - Ví dụ: Với $N_b = 4$:
        - khóa vòng đầu tiên là W[0], W[1], W[2], W[3] **(W[0, 3])**
        - khóa vòng thứ hai là W[4], W[5], W[6], W[7] **(W[4, 7])** v.v.
- **Tổng số bit của các khóa vòng** bằng chiều dài khối nhân với số vòng cộng thêm 1 (vì có vòng khởi tạo).
    - **VD:** Với AES-128 (khối 128 bit, 10 vòng): Tổng số bit = 128 × (10 + 1) = 128 × 11 = 1408 bit.
- **Thuật toán Key Expansion/ Hàm mở rộng khóa**
    - Khóa mở rộng (Expanded Key) được tạo ra từ khóa mã hóa (Cipher Key) bằng thuật toán **Key Expansion**. Thuật toán này có hai phiên bản, tùy thuộc vào $N_k$:

**# Trường hợp $N_k \leq 6$: $N_k = 4$ (AES-128), $N_k = 6$ (AES-192).**

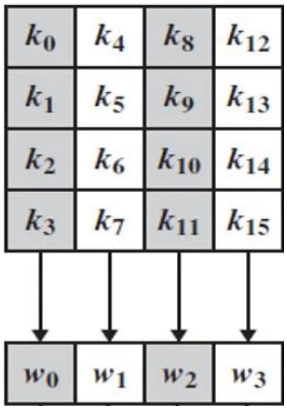
```
KeyExpansion(byte Key[4*Nk], word W[Nb * (Nr + 1)])
{
  for(i = 0; i < Nk; i++)
    W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);

  for(i = Nk; i < Nb*(Nr+1); i++) {
    temp = W[i-1];

    if (i % Nk == 0)
      temp = ByteSub(RotByte(temp)) ^ Rcon[i/Nk];

    W[i] = W[i-Nk] ^ temp;
  }
}
```

- **Giải thích:**




- **i. Khởi tạo các từ đầu tiên (0-3) từ khóa gốc**

**for(i = 0; i < $N_k$; i++)**
**W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);**

- Mỗi W[i] là **một từ (word)** gồm **4 byte**.
  - W[0] = {K0, K1, K2, K3} ← Key[0..3]
  - W[1] = {K4, K5, K6, K7} ← Key[4..7]
  - W[2] = {K8, K9, K10, K11} ← Key[8..11]
  - W[3] = {K12, K13, K14, K15} ← Key[12..15]

- **ii. Sinh các từ còn lại**

**for(i = $N_k$; i < $N_b$*($N_r$+1); i++) {**

- Duyệt từ W[$N_k$] đến W[$N_b$*($N_r$+1)-1].
  Với AES-128: **i = 4 → 43** (vì $N_b$*($N_r$+1) = 4*11 = 44 từ khóa con).

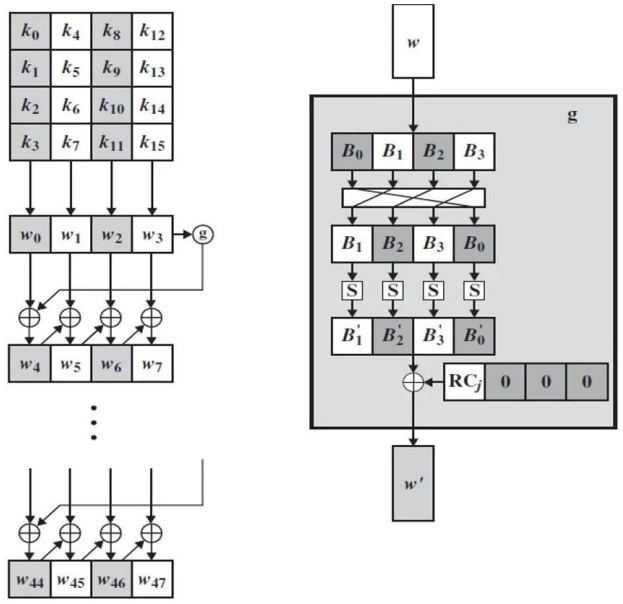
**temp = W[i-1];**

- Lấy từ khóa ngay trước W[i] để dùng tạo khóa tiếp theo.

**if (i % $N_k$ == 0)**
**temp = ByteSub(RotByte(temp)) ^ Rcon[i/$N_k$];**

  - Khi i là **bội số của $N_k$** (vị trí đầu của mỗi nhóm $N_k$ từ mới), thì:
  - **a. RotByte(temp)**
    - **Xoay** vòng **1 byte theo quy tắc:** [a0, a1, a2, a3] → [a1, a2, a3, a0]
    - Giúp tạo sự khác biệt giữa các khóa.
  - **b. ByteSub(...)**
    - Áp dụng hàm **S-box** cho từng byte (hàm phi tuyến), làm tăng tính bảo mật.
    - Đây là cùng hàm dùng trong SubBytes của AES.

- **c. ^ Rcon[i/$N_k$]**
  - XOR với Rcon[i/$N_k$]:
  - Rcon là hằng số vòng: dạng Rcon[1] = {01,00,00,00}, Rcon[2] = {02,00,00,00},...
  - Giúp đảm bảo mỗi vòng có khóa khác nhau.




- **Rcon** là một word (4 bytes) định nghĩa là
- Rcon[j] = (RC[j], 0, 0, 0),
- RC[i], là các thành phần trong GF ($2^8$)
  với **RC[1] = 1 (tức là 01 trong hệ 16),**
  RC[2] = x (tức là 02 trong hệ 16)...
- **RC[j] = 2*RC[j-1]** với phép nhân được định nghĩa trên trường GF($2^8$).

- Biểu diễn các RC[j] **trên hệ 16,** ta được bảng bên dưới

| Round | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| RCON | 01 | 02 | 04 | 08 | 10 | 20 | 40 | 80 | 1B | 36 |
| | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

- **iii. Tạo từ khóa mới bằng XOR**

**W[i] = W[i-$N_k$] ^ temp;**

  - XOR giữa:
    - **W[i-$N_k$]:** Từ khóa $N_k$ trước đó
    - **temp:** là từ vừa biến đổi từ W[i-1]
  - **→ Sinh ra W[i], từ khóa mới.**

**# Trường hợp $N_k$>6: $N_k$ = 8 (AES-256):**

```
KeyExpansion(byte Key[4*Nk], word W[Nb*(Nr+1)])
{
  for(i = 0; i < Nk; i++)
    W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);

  for(i = Nk; i < Nb*(Nr+1); i++) {
    temp = W[i-1];
    if(i % Nk == 0)
      temp = SubByte(RotByte(temp)) ^ Rcon[i/Nk];
    else if(i % Nk == 4)
      temp = SubByte(temp);
    W[i] = W[i-Nk] ^ temp;
  }
}
```

## 10. Ví dụ về mã hóa đối xứng AES 128 bit

### a/ Sinh khóa:

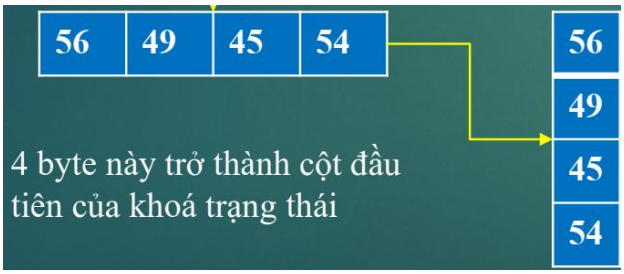
- Cho Key đầu vào/key mã hóa (cipher key): **VIETNAMUKRAINE12**

| V | I | E | T | N | A | M | U | K | R | A | I | N | E | 1 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- **V** Trong hệ 16 là 56
  Trong hệ nhị phân là 01010110
- Ta đưa cipher key về dạng hệ 16.

| 56 | 49 | 45 | 54 | 4E | 41 | 4D | 55 | 4B | 52 | 41 | 49 | 4E | 45 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

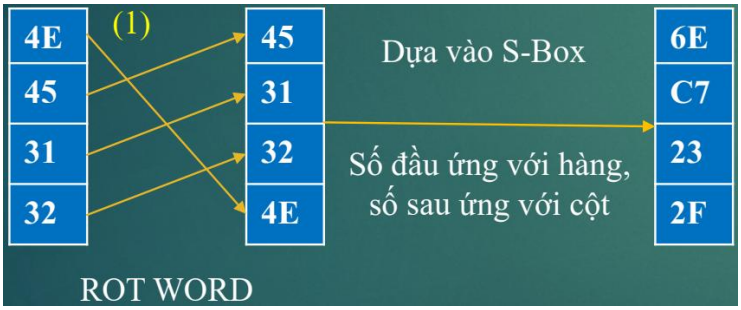
- **Lần lượt lấy 4 byte của key lập thành 1 cột của ma trận khóa.**



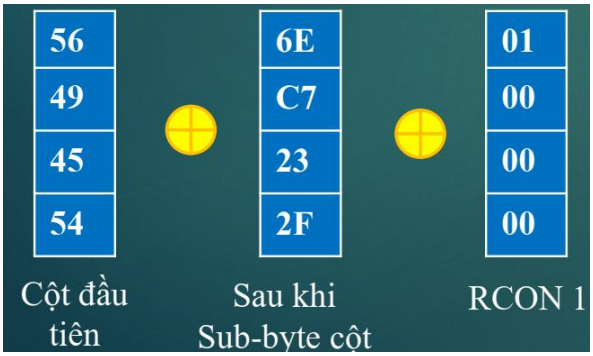

- Cuối cùng ta được ma trận key **(cũng chính là khóa sử dụng cho vòng 0 – $K_0$).**

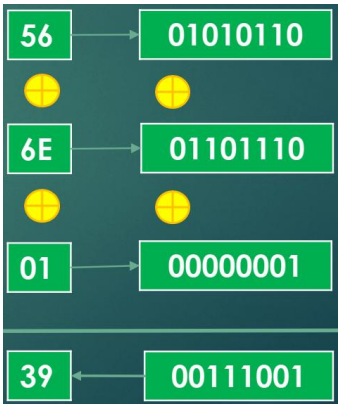
| 56 | 4E | 4B | 4E |
|---|---|---|---|
| 49 | 41 | 52 | 45 |
| 45 | 4D | 41 | 31 |
| 54 | 55 | 49 | 32 |
| W[0] | W[1] | W[2] | W[3] |

- **Sinh khóa mở rộng → khóa vòng/ khóa con**
  - Ta sử dụng ma trận khóa trước đó, tức là ma trận khóa $K_0$, để thực hiện sinh khóa $K_1$.
  - Đầu tiên thực hiện sinh cột đầu tiên của khóa $K_1$, tương ứng với việc ta đang thực hiện trên W[4], chính là cột đầu tiên của khóa $K_1$, mà vì chỉ số của W hiện là 4 là bội của $N_k$ ($N_k = 4$). Với điều kiện chỉ số của W là bội của $N_k$ (như mô tả thuật toán ở trên), ta thực hiện các bước sau:
    - Từ khóa $K_0$, **Lấy cột cuối của khoá và thực hiện ROT WORD**, nghĩa là $[a_0, a_1, a_2, a_3] \rightarrow [a_1, a_2, a_3, a_0]$
    - Sau đó **thực hiện subtitute byte** dựa vào S-box chuyển đổi.




    - Tiếp theo, lấy KQ vừa có thực hiện **XOR với RCON[1] = [1,0,0,0] và W[0],** chính là cột đầu tiên của khóa $K_0$.

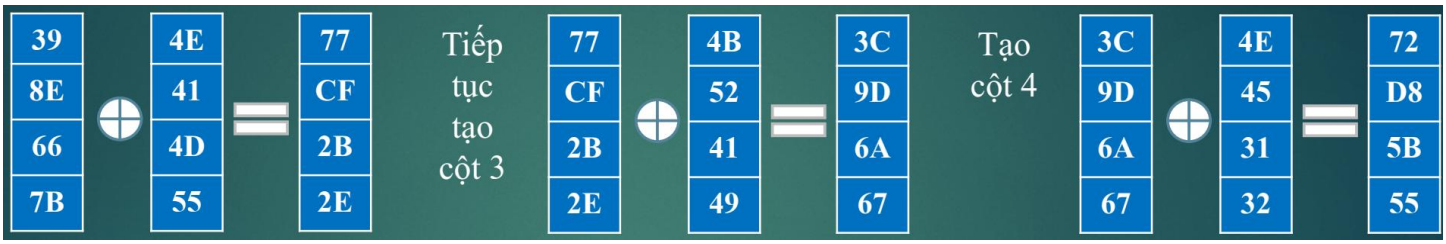


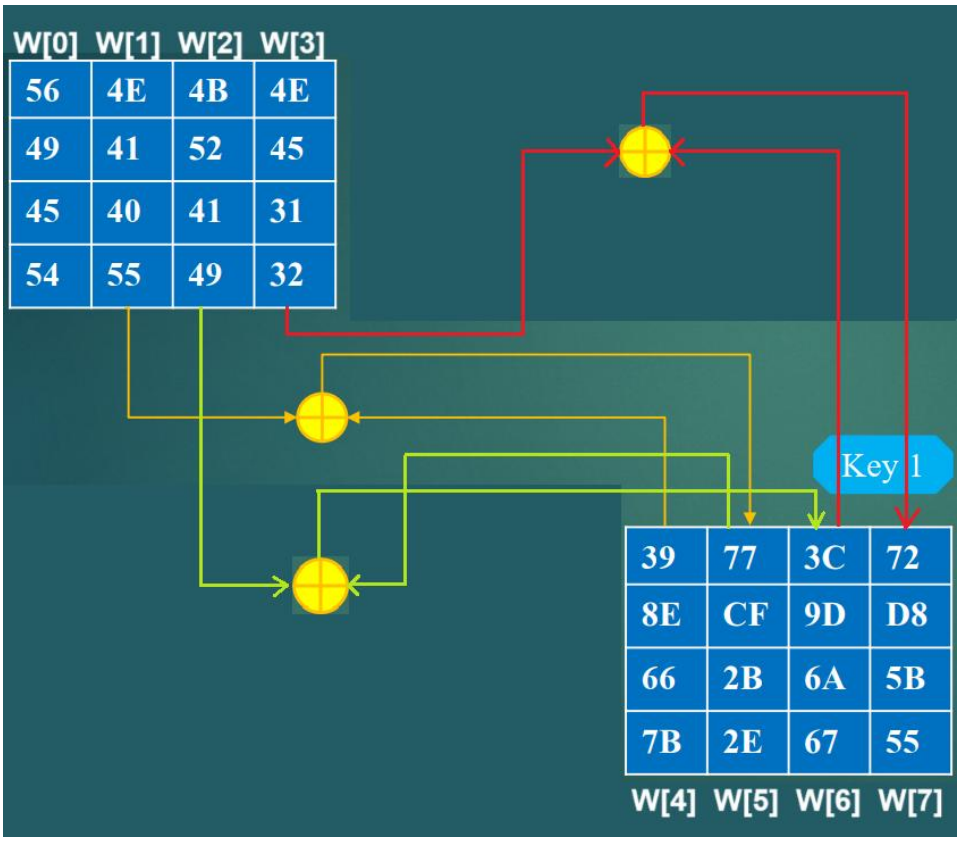

- Thực hiện với các dòng còn lại, ta được kết quả của cột đầu tiên của $K_1$, chính là W[4], đó là:


- Các cột tiếp theo, tương ứng với W[5], W[6], W[7], ta chỉ thực hiện XOR theo cú pháp sau:

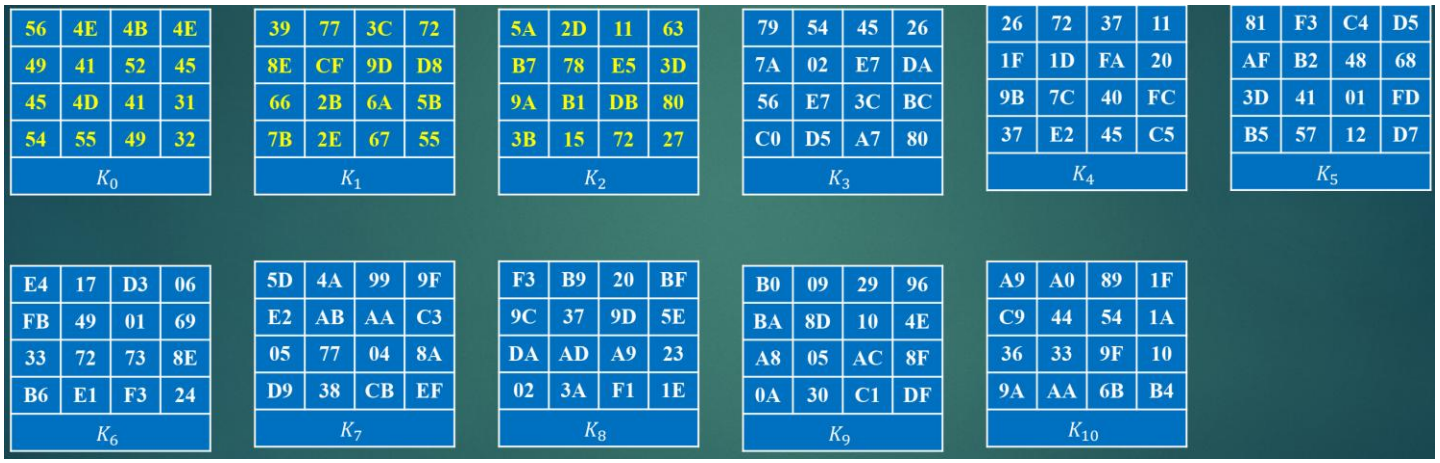
**$W[i] = W[i\text{-}N_k] \oplus W[i\text{-}1];$**




- **Cuối cùng, ta được ma trận $K_1$ như hình dưới:**




- **Thiết lập $K_2$ từ $K_1$**: Lấy cột cuối của $K_1$ và thực hiện tương tự như trên.
- **Sau tất cả 10 lần,** việc sinh khóa dừng lại, và ta thu được một ma trận khóa mở rộng, trong đó có 11 ma trận khóa con sinh từ khóa trạng thái từ $K_0$ → $K_{10}$.



**b/ Mã hóa tin nhắn**

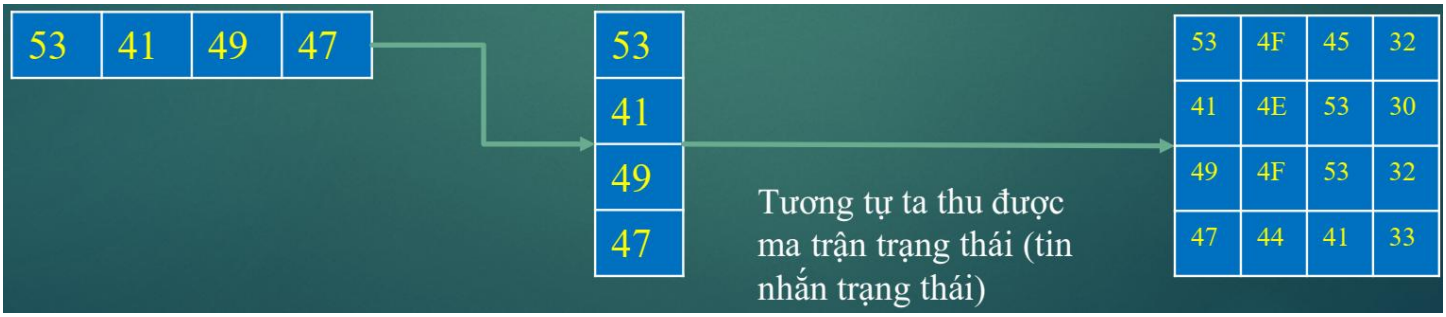
- **Quá trình mã hóa đã mô tả chi tiết ở các phần trên.**
- Giả sử tin nhắn cần mã hóa như sau:

| S | A | I | G | O | N | O | D | E | S | S | A | 2 | 0 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

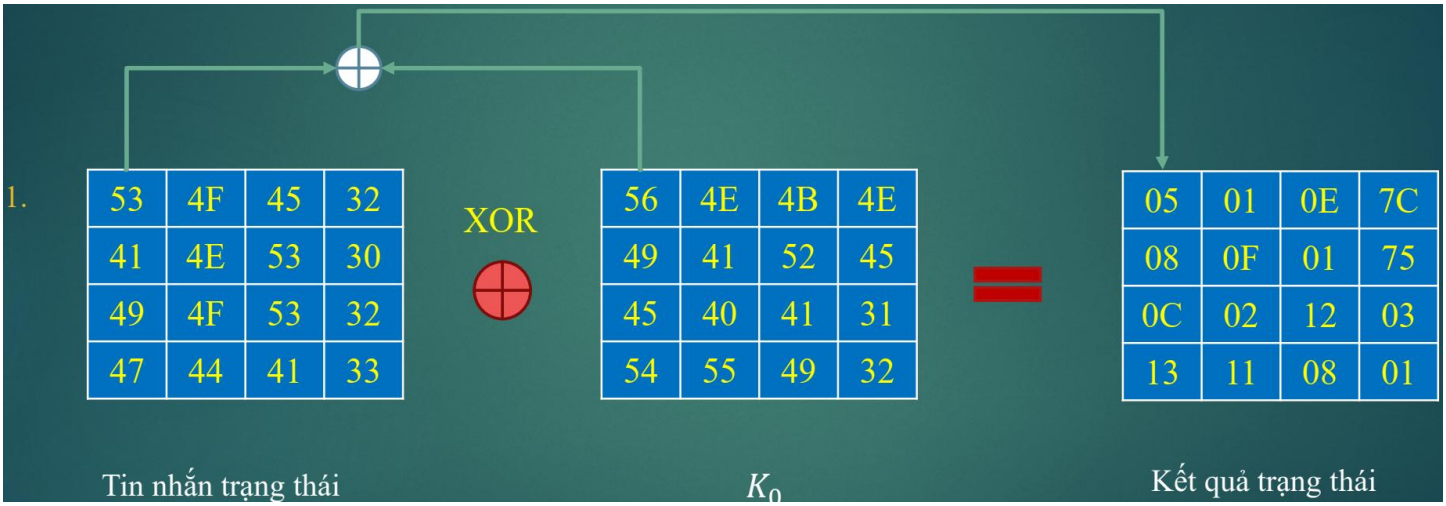
- Tương tự như khoá chuyển sang hệ 16


- Ta thu được:

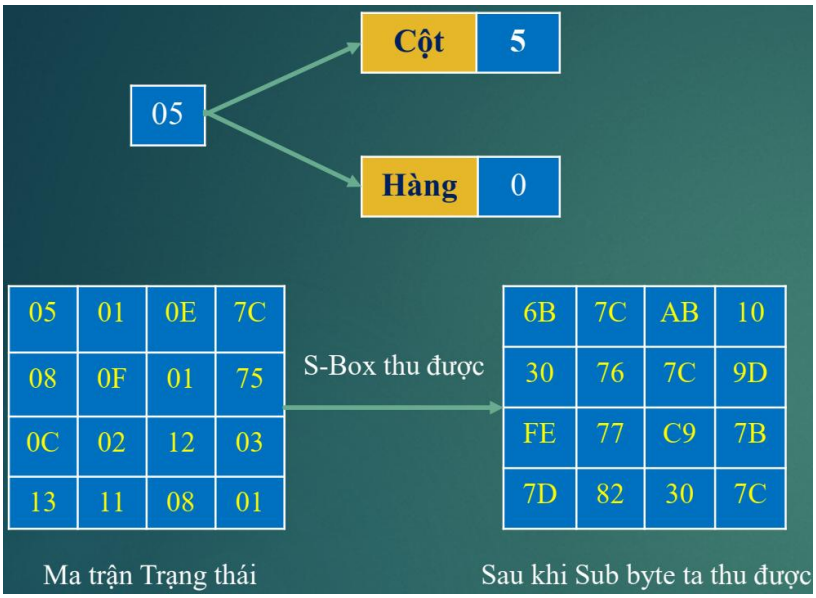
| 53 | 41 | 49 | 47 | 4F | 4E | 4F | 44 | 45 | 53 | 53 | 41 | 32 | 30 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



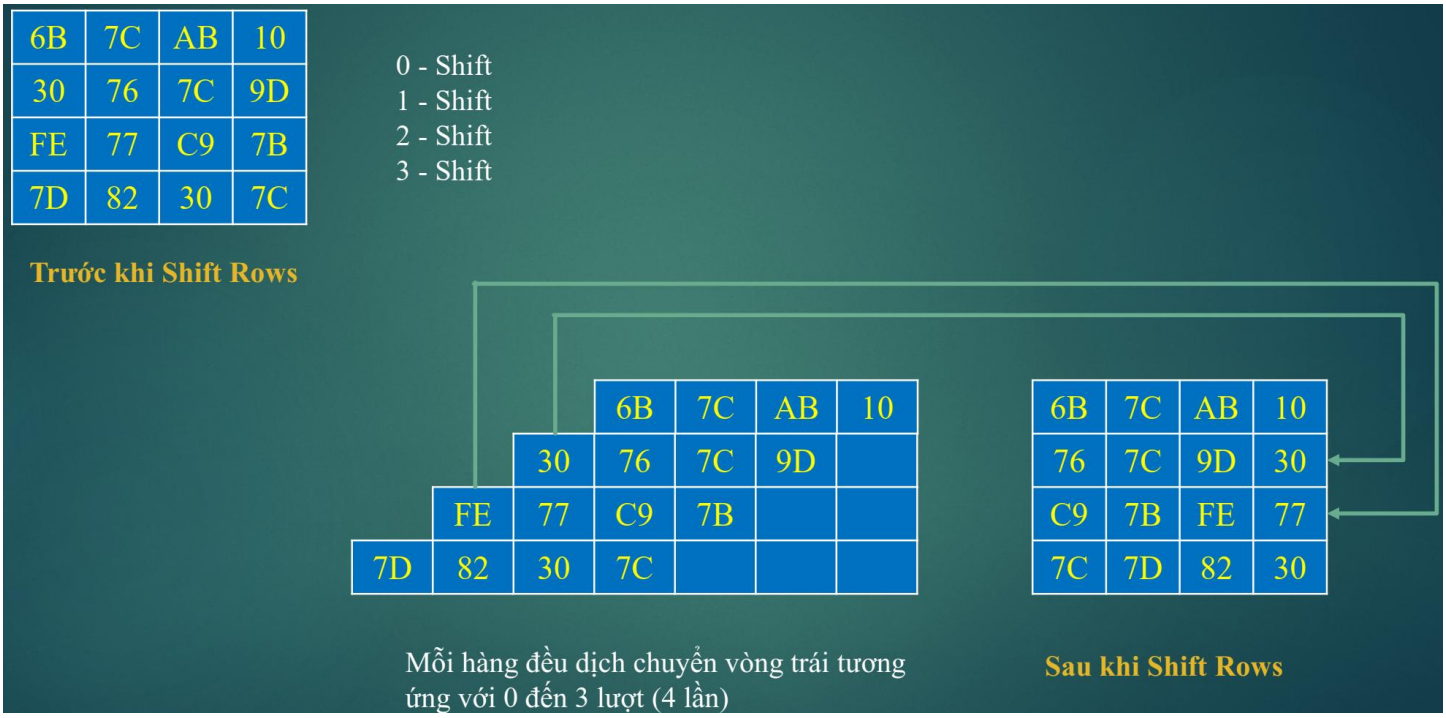

- **Add Round Key:**



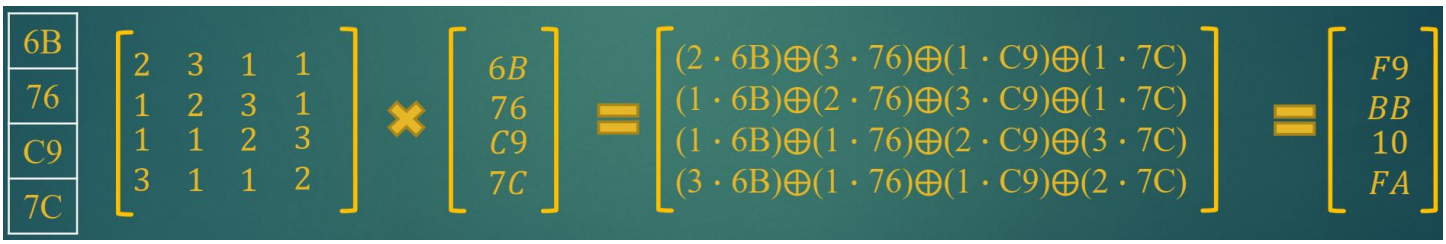
❖ **Sub byte**

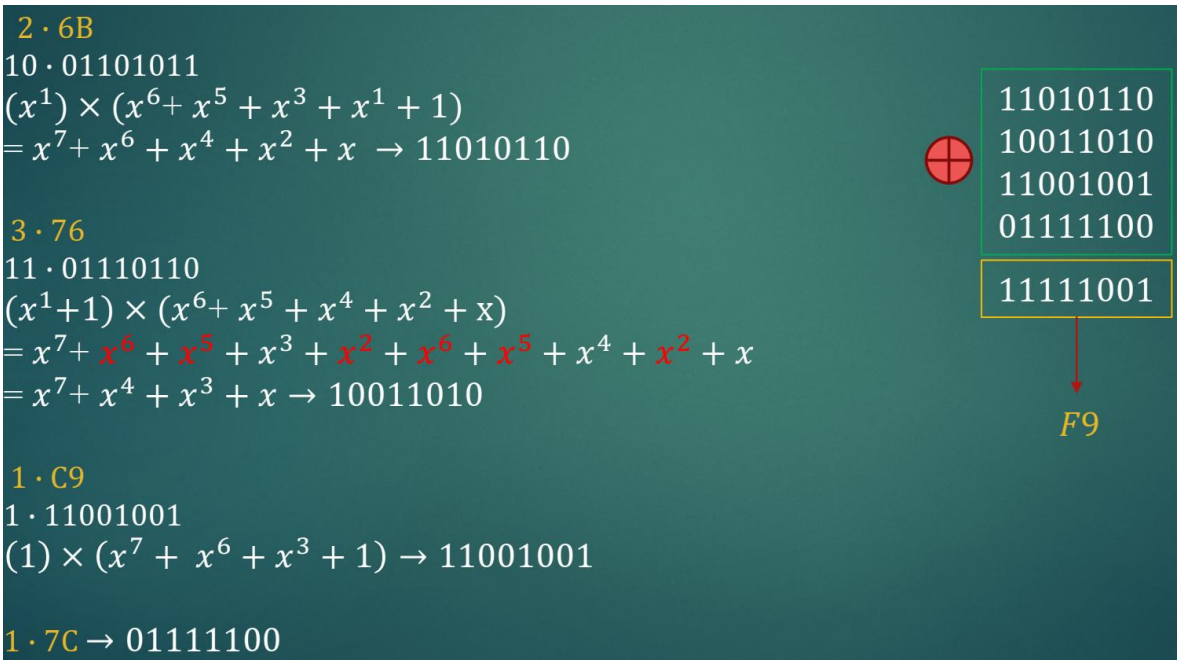



❖ **Shift Rows**




❖ **Mix Columns:** Xét cột đầu tiên của ma trận sau khi thực hiện Shift Rows.






- **Thực hiện tương tự với các cột, dòng còn lại, ta thu được ma trận Mix Column**

❖ **Tiếp tục thực hiện 4 phép biến đổi với các vòng còn lại, từ vòng 2-9. Tuy nhiên ở vòng 10 chỉ thực hiện 3 phép Sub-byte, ShiftRows, và AddRoundKey.**

**Tại sao vòng cuối của thuật toán AES không áp dụng phép biến đổi MixColumns?**

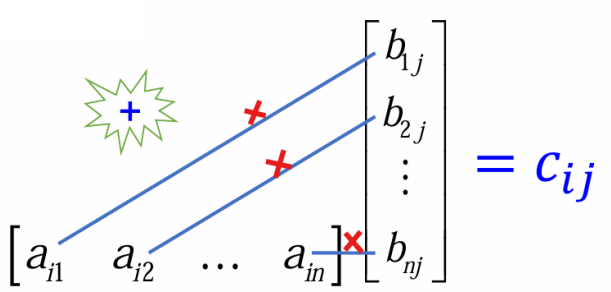
- Giữ tính đối xứng của thuật toán mã hóa và giải mã
  - **Mã hóa:** Initial Round (AddRoundKey) → Main Rounds (4 bước) → Final Round (3 bước: SubBytes, ShiftRow, AddRoundKey).
  - **Giải mã:** Initial Round (Inverse AddRoundKey) → Main Rounds (4 bước đảo ngược: Inverse ShiftRow, Inverse SubBytes, Inverse MixColumns, Inverse AddRoundKey) → Final Round (3 bước: Inverse ShiftRow, Inverse SubBytes, Inverse AddRoundKey).
- Nếu vòng cuối của mã hóa áp dụng MixColumns, thì vòng đầu của giải mã (tương ứng với vòng cuối của mã hóa) sẽ phải áp dụng **Inverse MixColumns**. Tuy nhiên, vòng đầu của giải mã thường bắt đầu bằng **Inverse AddRoundKey**, và việc thêm Inverse MixColumns ở đây sẽ làm phức tạp quá trình giải mã, đặc biệt là khi triển khai phần cứng hoặc phần mềm.
- **Phép MixColumns là bước phức tạp nhất** trong AES về mặt tính toán, vì nó yêu cầu thực hiện các phép nhân trong trường GF($2^8$) và các phép XOR. Việc bỏ MixColumns ở vòng cuối giúp giảm tải tính toán, đặc biệt trong các hệ thống có tài nguyên hạn chế (như phần cứng nhúng hoặc thiết bị IoT).
- **MixColumns** đóng vai trò quan trọng trong việc tăng tính khuếch tán (diffusion), tức là đảm bảo rằng một thay đổi nhỏ trong dữ liệu đầu vào (plaintext) sẽ ảnh hưởng đến toàn bộ dữ liệu đầu ra (ciphertext) sau nhiều vòng.
  - Tuy nhiên, ở vòng cuối, mục tiêu chính không phải là khuếch tán thêm, mà là hoàn tất quá trình mã hóa để tạo ra ciphertext. Các vòng trước đã thực hiện đủ khuếch tán, và việc bỏ MixColumns ở vòng cuối không làm giảm đáng kể tính bảo mật

**Phép nhân hai ma trận:**

Cho hai ma trận $\boldsymbol{A} = [a_{ij}]_{m\times n}$ và $\boldsymbol{B} = [b_{ij}]_{n\times p}$ (với **số cột của ma trận A bằng số dòng của ma trận B**).

Ta định nghĩa tích của A với B là ma trận $\boldsymbol{C} = [c_{ij}]_{m\times p}$ trong đó các phần tử $c_{ij}$ được xác định như sau:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}.$$




**Ví dụ:**

$$\begin{bmatrix} 2 & -1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 4 \end{bmatrix} = [2.1 + (-1).0 + 3.4] = [14]$$

# ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [END OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |