Lecture A4:

AES: The Advanced Encryption Standard

4th Year Course- CCSIT, UoA

Lecture goals

- To review the overall structure of AES.
- To focus particularly on the four steps used in each round of AES: (1) substitution, (2) shift rows, (3) mix columns, and (4) add round key.
- To go through the details of how the encryption key is expanded to yield the round keys.

Salient Features of AES (1)

- AES is a block cipher with a block length of 128 bits.
- AES allows for three different key lengths: 128, 192, or 256 bits. Our discussion will assume that the key length is 128 bits.
- Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.
- Except for the last round in each case, all other rounds are identical.
- Each round of processing consists of one single-byte based substitution step, followed by what is known as a row-wise permutation step, followed by a "column-based" substitution step, followed by the addition of the round key.
- To appreciate the processing steps used in a single round, it is best to think of a 128-bit block as consisting of a 4 × 4 matrix of bytes, arranged as shown in the next slide:

Salient Features of AES (2)

- Therefore, the first four bytes of a 128-bit input block occupy the first column in the 4 × 4 matrix of bytes. The next four bytes occupy the second column, and so on.

$$\begin{bmatrix} byte_0 & byte_4 & byte_8 & byte_{12} \\ byte_1 & byte_5 & byte_9 & byte_{13} \\ byte_2 & byte_6 & byte_{10} & byte_{14} \\ byte_3 & byte_7 & byte_{11} & byte_{15} \end{bmatrix}$$

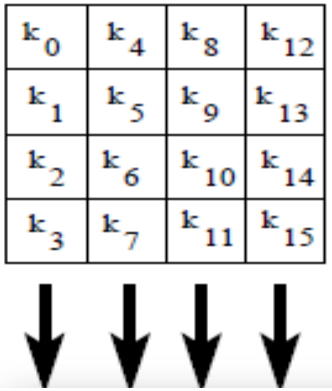
- The 4 × 4 matrix of bytes is referred to as the **state array**.
- AES also has the notion of a **word**. A word consists of four bytes, that is 32 bits. Therefore, each column of the state array is a word, as is each row.
- Each round of processing works on the input state array and produces an output state array.
- The output state array produced by the last round is rearranged into a 128-bit output block.

Salient Features of AES (3)

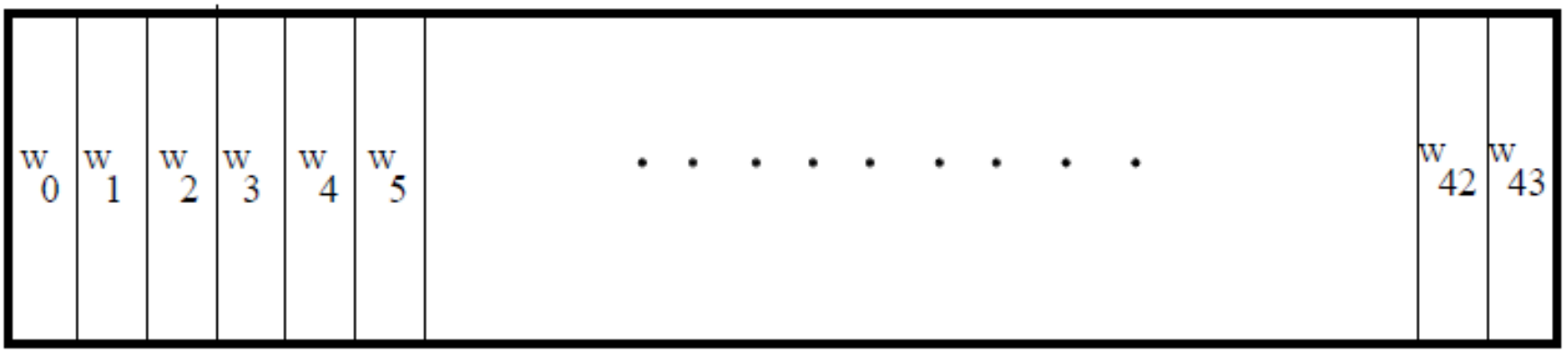
- Unlike DES, the decryption algorithm differs substantially from the encryption algorithm. Although, overall, the same steps are used in encryption and decryption, the order in which the steps are carried out is different.
- AES, notified by NIST as a standard in 2001, is a slight variation of the Rijndael cipher invented by two Belgian cryptographers Joan Daemen and Vincent Rijmen.
- Whereas AES requires the block size to be 128 bits, the original Rijndael cipher works with any block (and key) size that is a multiple of 32 as long as it exceeds 128. The state array and the key array for variable block and key sizes still have only four rows. The number of columns is depends on size of the block and the size of the key.
- DES is based on the Feistel network. On the other hand, what AES uses is a substitution-permutation network in a more general sense. The nature of substitutions and permutations in AES allows for fast software implementations of the algorithm.

The Encryption Key and Its Expansion

- Assuming a 128-bit key, the key is also arranged in the form of a matrix of 4 × 4 bytes. As with the input block, the first word from the key fills the first column of the matrix, and so on.
- The four column words of the key matrix are expanded into a schedule of 44 words. (As to how exactly this is done, we will explain that later.) Each round consumes four words from the key schedule.



- This figure depicts the arrangement of the encryption key in the form of 4-byte words and the expansion of the key into a key schedule consisting of 44 4-byte words.

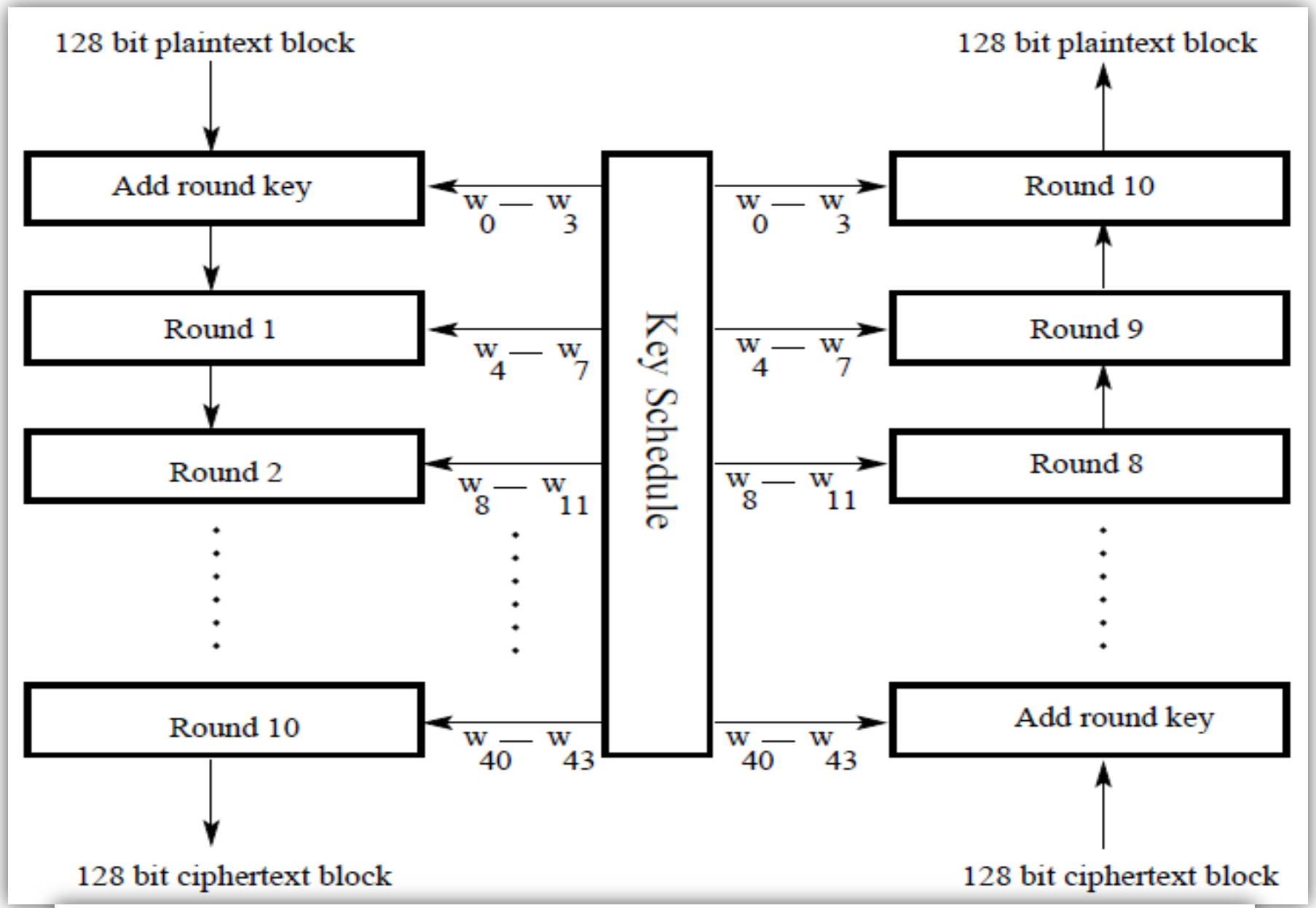


The Overall Structure of AES (1)

- The overall structure of AES encryption/decryption is shown in the next figure (slide 9).
- The number of rounds shown in the figure, 10, is for the case when the encryption key is 128 bit long. (As mentioned earlier, the number of rounds is 12 when the key is 192 bits, and 14 when the key is 256.)
- Before any round-based processing for encryption can begin, the input state array is XORed with the first four words of the key schedule. The same thing happens during decryption — except that now we XOR the ciphertext state array with the last four words of the key schedule.

The Overall Structure of AES (2)

- For encryption, each round consists of the following four steps: 1) Substitute bytes, 2) Shift rows, 3) Mix columns, and 4) Add round key. The last step consists of XORing the output of the previous three steps with four words from the key schedule.
- For decryption, each round consists of the following four steps: 1) Inverse shift rows, 2) Inverse substitute bytes, 3) Add round key, and 4) Inverse mix columns. The third step consists of XORing the output of the previous two steps with four words from the key schedule.
- The last round for encryption does not involve the "Mix columns" step. The last round for decryption does not involve the "Inverse mix columns" step.



AES Encryption AES Decryption

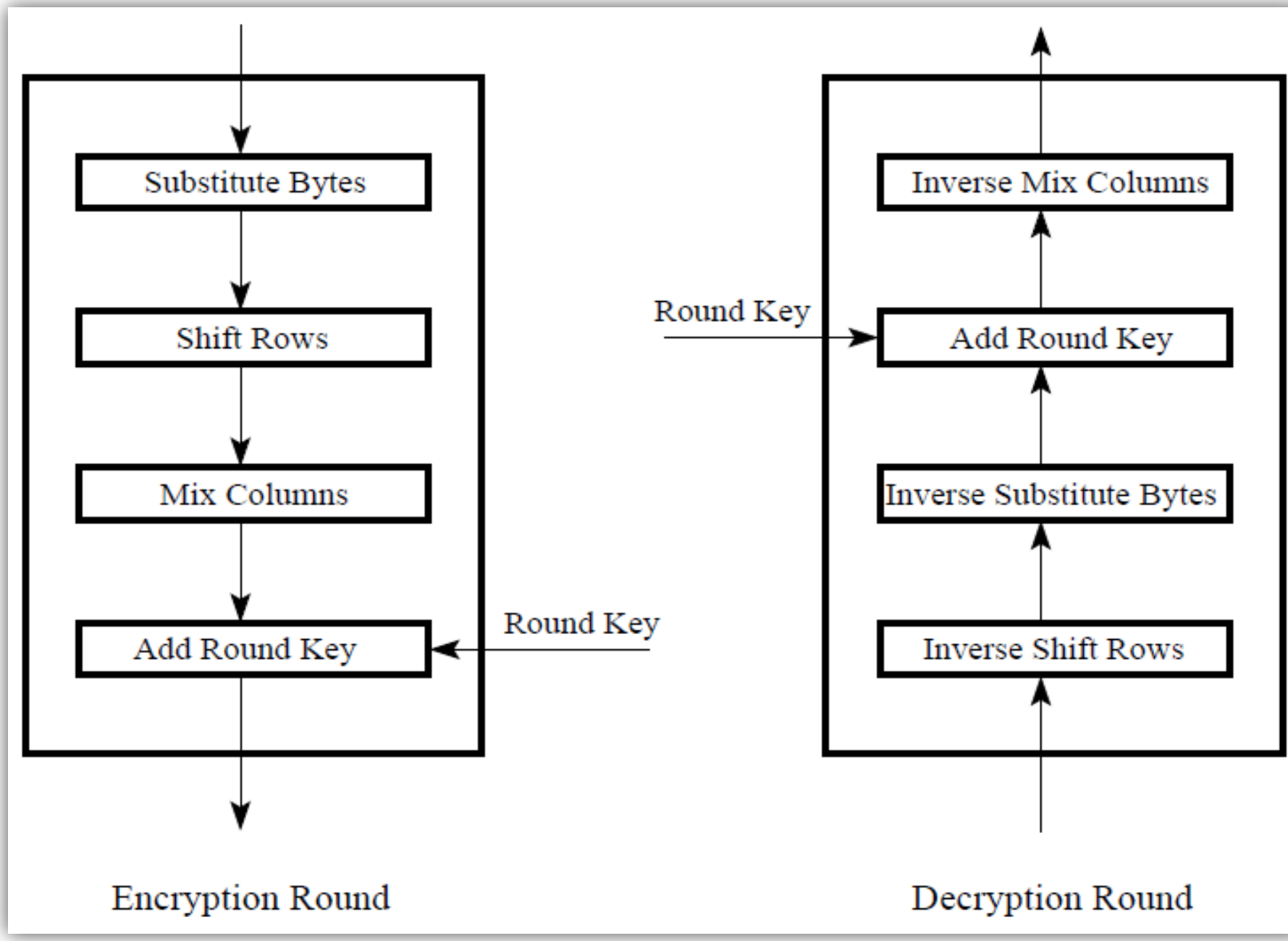
The Four Steps in Each Round of Processing (1)

The next figure (slide 12) shows the different steps that are carried out in each round except the last one.

- STEP 1: (called SubBytes for byte-by-byte substitution during the forward process) (The corresponding substitution step used during decryption is called InvSubBytes.)
  - This step consists of using a 16 × 16 lookup table to find a replacement byte for a given byte in the input state array.
  - The entries in the lookup table are created by using the notions of multiplicative inverses in GF (2^8) and bit scrambling to destroy the bit-level correlations inside each byte.
- STEP 2: (called ShiftRows for shifting the rows of the state array during the forward process) (The corresponding transformation during decryption is denoted InvShiftRows for Inverse Shift-Row Transformation.)
  - The goal of this transformation is to scramble the byte order inside each 128-bit block.

The Four Steps in Each Round of Processing (2)

- STEP 3: (called MixColumns for mixing up of the bytes in each column separately during the forward process) (The corresponding transformation during decryption is denoted InvMixColumns and stands for inverse mix column transformation.) The goal is here is to further scramble up the 128-bit input block.
  - The shift rows step along with the mix column step causes each bit of the ciphertext to depend on every bit of the plaintext after 10 rounds of processing.
  - Recall the avalanche efect from our discussion on DES. In DES, one bit of plaintext affected roughly 31 bits of ciphertext.
  - But now we want each bit of the plaintext to affect every bit of the ciphertext in a block of 128 bits.
- STEP 4: (called AddRoundKey for adding the round key to the output of the previous step during the forward process) (The corresponding step during decryption is denoted InvAddRoundKey for inverse add round key transformation.)


Substitute Bytes
Shift Rows
Mix Columns
Add Round Key
Round Key
Encryption Round
Inverse Mix Columns
Round Key
Add Round Key
Inverse Substitute Bytes
Inverse Shift Rows
Decryption Round

The Substitute Bytes Step

- This is a byte-by-byte substitution and the substitution byte for each input byte is found by using the same lookup table.
- The size of the lookup table is 16 × 16.
- To find the substitute byte for a given input byte, we divide the input byte into two 4-bit patterns, each yielding an integer value between 0 and 15. (We can represent these by their hex values 0 through F.) One of the hex values is used as a row index and the other as a column index for reaching into the 16 × 16 lookup table.
- The entries in the lookup table are constructed by a combination of GF (2^8) arithmetic and bit mangling.
- The goal of the substitution step is to reduce the correlation between input bits and output bits (at the byte level). The bit mangling part of the substitution step ensures that the substitution cannot be described in the form of evaluating a simple mathematical function.

Construction of the 16 × 16 Lookup Table for the *SubBytes* Step (1)

- We first fill each cell of the 16 × 16 table with the byte obtained by joining together its row index and the column index.
- For example, for the cell located at row 3 and column A, we place 3A in the cell. So at this point the table will look like

	0	1	2	3	4	5	6	7	8	9	...
0	00	01	02	03	04	05	06	07	08	09	...
1	10	11	12	13	14	15	16	17	18	19	...
2	20	21	22	23	24	25	26	27	28	29	...
	.....										
	.....										

- We next replace the value in each cell by its multiplicative inverse in GF (2^8) based on the irreducible polynomial $x^8+x^4+x^3+x+1$. The value 00 is replaced by itself since this element has no multiplicative inverse.

Construction of the 16 × 16 Lookup Table for the *SubBytes* Step (2)

- Let's represent a byte stored in each cell of the table by $b_7b_6b_5b_4b_3b_2b_1b_0$ where b_7 is the MSB and b_0 the LSB. For example, the byte stored in the cell (9, 5) of the above table is the multiplicative inverse of 0*x*95, which is 0*x*8*A*. Therefore, at this point, the bit pattern stored in the cell with row index 9 and column index 5 is 10001010, implying that b_7 is 1 and b_0 is 0.

❖ Question: Verify the fact that the MI of 0*x*95 is indeed 0*x*8*A*. [The polynomial representation of 0*x*95 (bit pattern: 10010101) is $x^7 + x^4 + x^2 + 1$, and the same for 0*x*8*A* (bit pattern: 10001010) is $x^7 + x^3 + x$. Now show that the product of these two polynomials modulo the polynomial $x^8 + x^4 + x^3 + x + 1$ is 1.]

- For bit mangling, we next apply the following transformation to each bit b_i of the byte stored in a cell of the lookup table:

$$b'_i = b_i \otimes b_{(i+4) \bmod 8} \otimes b_{(i+5) \bmod 8} \otimes b_{(i+6) \bmod 8} \otimes b_{(i+7) \bmod 8} \otimes c_i$$

where c_i is the i^{th} bit of a specially designated byte c whose hex value is *Ox*63. ($c_7c_6c_5c_4c_3c_2c_1c_0 = 01100011$)

Construction of the 16 × 16 Lookup Table for the *SubBytes* Step (3)

- The above bit-mangling step can be better visualized as the following vector-matrix operation. Note that all of the additions in the product of the matrix and the vector are actually XOR operations.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

- The 16 × 16 table created in this manner is called the S-Box. The S-Box is the same for all the bytes in the state array.
- The extra bit-mangling introduced by the transformation shown above is meant to break the correlation between the bits before the substitution and the bits after the substitution while maintaining a relationship between the input bytes and the output bytes that can be described by algebraic equations.

Construction of the 16 × 16 Lookup Table for the *SubBytes* Step (4)

- This byte-by-byte substitution step is reversed during decryption, meaning that you first lookup the value in the decryption S-box and then you take its multiplicative inverse in *GF* (2^8).
- The 16×16 lookup table for decryption is constructed by starting out in the same manner as for the encryption lookup table. That is, you place in each cell the byte constructed by joining the row index with the column index. Then, for bit mangling, you carry out the following bit-level transformation in each cell of the table:

$$b'_i = b_{(i+2) \bmod 8} \otimes b_{(i+5) \bmod 8} \otimes b_{(i+7) \bmod 8} \otimes d_i$$

where d_i is the i^{th} bit of a specially designated byte d whose hex value is $Ox05$. ($d_7d_6d_5d_4d_3d_2d_1d_0 = 00000101$) Finally, you replace the byte in the cell by its multiplicative inverse in *GF* (2^8).

- The bytes c and d are chosen so that the S-box has no fixed points. That is, we do not want *S_box* (a) = a. Neither do we want *S_box* (a) = ā where ā is the bitwise complement of a.