Lecture 8: AES: The Advanced Encryption Standard

Lecture Notes on "Introduction to Computer Security"

by Avi Kak (kak@purdue.edu)

February 8, 2007

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

- 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 follows:

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

- 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.

- The 4×4 matrix of bytes is referred to as the **state array**.

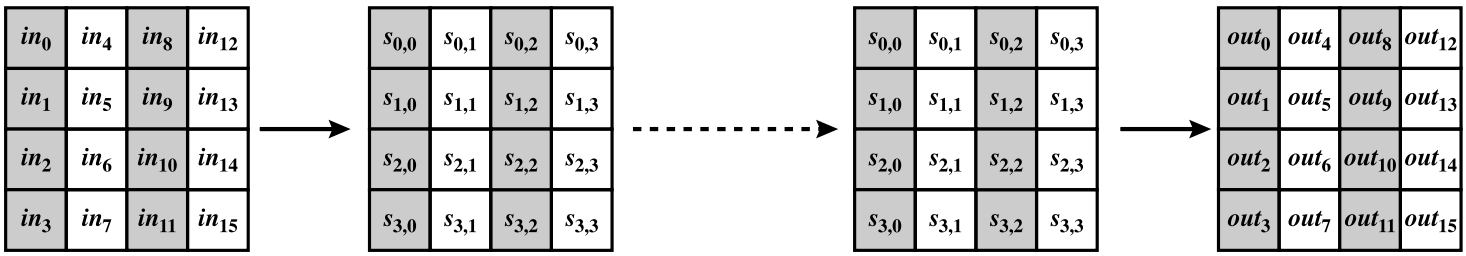
- AES also has the notion of a **word**. A word consists of four bytes, that is 32 bits. Therefore, each column of the state array is a word, as is each row.

- 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.

- Unlike DES, the decryption algorithm differs substantially from the encryption algorithm. Although the encryption and the decryption algorithms are structurally the same (meaning that the flow of logic is the same), nonetheless the two differ with regard to the details of the various transformation used.

- The figure below shows the propagation of the state array from round to round in AES.



This figure is from Chapter 5 of Stallings: "Cryptography and Network Security", Fourth Edition

The Encryption Key and Its Expansion

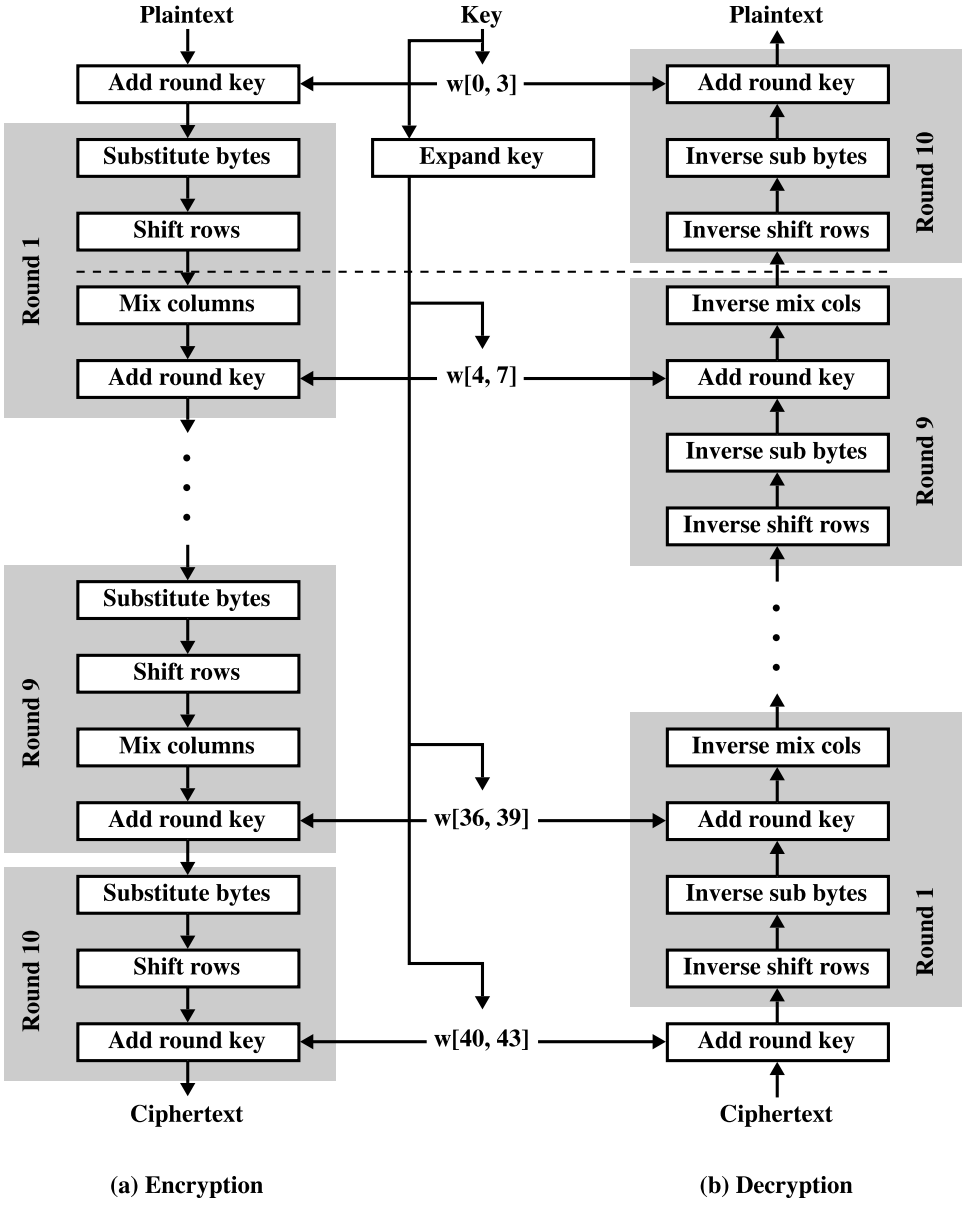
- Assuming a 128-bit key, the key is also arranged in the form of a matrix of 4×4 bytes. As with the input block, the first word from the key fills the first column of the matrix, and so on.

- The four column words of the key matrix are expanded into a schedule of 44 words. Each round consumes 4 key words.

- The figure below 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.

This figure is from Chapter 5 of Stallings: "Cryptography and Network Security", Fourth Edition

The Overall Structure of AES



This figure is from Chapter 5 of Stallings: "Cryptography and Network Security", Fourth Edition

Each Round of Processing Consists of Four Steps

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.

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 effect 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.)

- The 128-bit key is expanded into a **key schedule** consists of 44 4-byte words.

- The first four words (consisting of 128 bits) of the key schedule are the same as the overall encryption key. These four words are used prior to any round-based processing; the input block is XOR'ed with these four words.

- From the rest of the words in the key schedule, consecutive groupings of four words are supplied to each of the 10 rounds. These serve as the **round key** for each of the rounds.

- Note that each round key is 128 bits in length.

The SubBytes Step in Each Round

- 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 value 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

- 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.

- 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 $0x95$, which is $0x8A$. 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.

- 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:

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

  where c_i is the i^{th} bit of a specially designated byte c whose hex value is $Ox63$. ($c_7c_6c_5c_4c_3c_2c_1c_0 \;=\; 01100011$)

- 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**.

- 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:

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

where d_i is the i^{th} bit of a specially designated byte d whose hex value is $Ox05$. ($d_7d_6d_5d_4d_3d_2d_1ddc_0 \quad = \quad 00000101$) 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 $S\_box(a) \quad = \quad a$. Neither do we want $S\_box(a) \quad = \quad \bar{a}$ where $\bar{a}$ is the bitwise complement of a.

ShiftRows Transformation

- This is where the matrix representation of the **state array** becomes important.

- The ShiftRows transformation consists of (i) not shifting the first row of the **state array** at all; (ii) circularly shifting the second row by one byte to the left; (iii) circularly shifting the third row by two bytes to the left; and (iv) circularly shifting the last row by three bytes to the left.

- This operation on the state array can be represented by

$$\begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \quad ===> \quad \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2.2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3.3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

- Recall again that the input block is written column-wise. That is the first four bytes of the input block fill the first column of the state array, the next four bytes the second column, etc. As a result, shifting the rows in the manner indicated scrambles up the byte order of the input block.

- For decryption, the corresponding step shifts the rows in exactly the opposite fashion. The first row is left unchanged, the second row is shifted to the right by one byte, the third row to the right by two bytes, and the last row to the right by three bytes, all shifts being circular.

$$\begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \quad ===> \quad \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.3} & s_{1,0} & s_{1,1} & s_{1,2} \\ s_{2.2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3.1} & s_{3,2} & s_{3,3} & s_{3,0} \end{bmatrix}$$

The MixColumns Transformation

- This step replaces each byte of a column by a function of all the bytes in the same column.

- More precisely, each byte in a column is replaced by two times the value of that byte, plus three times the next byte, plus the byte that comes next, plus the byte that comes next. The word 'next' means the byte in the row below; the meaning of 'next' is circular in the same column.

- For the bytes in the **first row** of the state array, this operation cane be stated as

$$s'_{0,j} \quad = \quad (2 \times s_{0,j}) \ \otimes \ (3 \times s_{1,j}) \ \otimes \ s_{2,j} \ \otimes \ s_{3,j}$$

- For the bytes in the **second row** of the state array, this operation cane be stated as

$$s'_{1,j} \quad = \quad s_{0,j} \ \otimes \ (2 \times s_{1,j}) \ \otimes \ (3 \times s_{2,j}) \ \otimes \ s_{3,j}$$

- For the bytes in the **third row** of the state array, this operation cane be stated as

$$s'_{2,j} \quad = \quad s_{0,j} \otimes s_{1,j} \otimes (2 \times s_{2,j}) \otimes (3 \times s_{3,j})$$

- And, for the bytes in the **fourth row** of the state array, this operation cane be stated as

$$s'_{3,j} \quad = \quad (3 \times s_{0,j}) \otimes s_{1,j} \otimes s_{2,j} \otimes (2 \times s_{3,j})$$

- More compactly, the column operations can be shown as

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 10 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0.0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1.0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2.0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3.0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

  where, on the left hand side, when a row of the leftmost matrix multiples a column of the state array matrix, additions involved are meant to be XOR operations.

- The corresponding transformation during decryption is given by

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} s_{0.0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1.0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2.0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3.0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0.0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1.0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2.0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3.0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Adding the Round Key

- The 128 bits of the state array are bitwise XOR'ed with the 128 bits of the round key.

- The **AES Key Expansion** algorithm is used to derive the 128-bit round key from the original 128-bit encryption key.

- In the same manner as the 128-bit input block is arranged in the form of a state array, the algorithm first arranges the 16 bytes of the encryption key in the form of a 4×4 array of bytes

$$\begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} w_0 & w_1 & w_2 & w_3 \end{bmatrix}$$

- The first four bytes of the encryption key constitute the word w_0, the next four bytes the word w_1, and so on.

- The algorithm subsequently expands the words $[w_0, w_1, w_2, w_3]$ into a 44-word **key schedule** that can be labeled

$$w_0,\ w_1,\ w_2,\ w_3, ..., \ w_{43}$$

- Of these, the words $[w_0, w_1, w_2, w_3]$ are bitwise XOR'ed with the input block **before** the round-based processing begins.

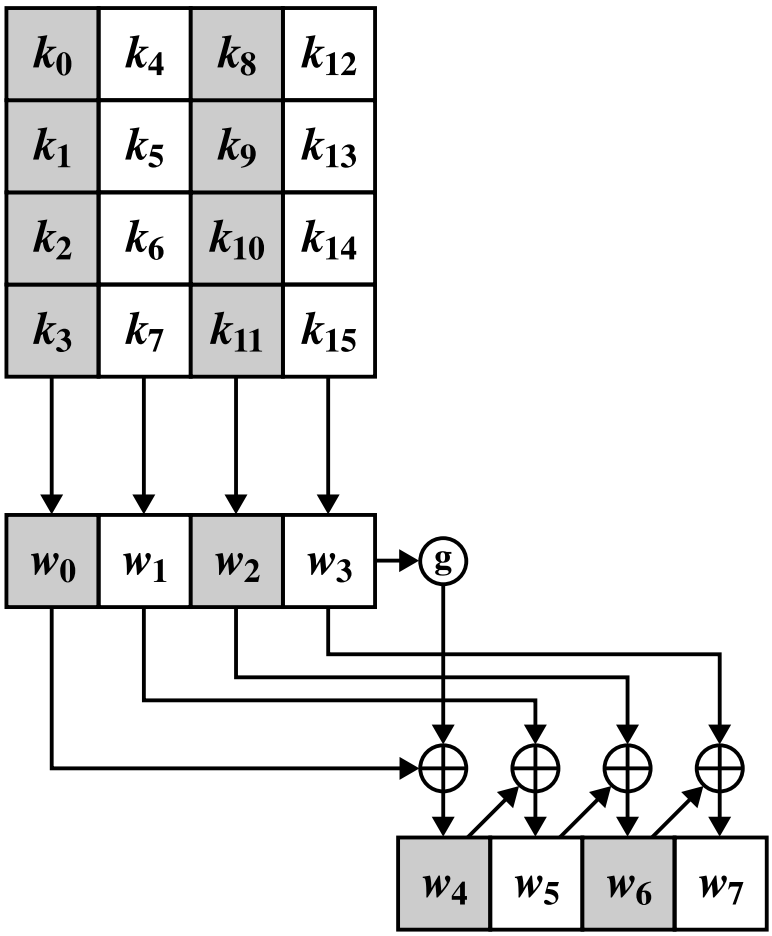
- The remaining 40 words of the key schedule are used **four words at a time** in each of the 10 rounds.

- The above two statements are also true for decryption. The first four words of the key schedule are bitwise XOR'ed with the 128-bit ciphertext block before any round-based processing begins. Subsequently, each of the four words in the remaining 40 words of the key schedule are used in each of the ten rounds of processing.

- Now comes the difficult part: How does the **Key Expansion Algorithm** expand four words w_0, w_1, w_2, w_3 into the 44 words $w_0, w_1, w_2, w_3, w_4, w_5, ..., w_{43}$?

- The key expansion will be explained with the help of the figure shown next.

- But first note that the key expansion takes place on a four-word to four-word basis, in the sense that each grouping of four words decides what the next grouping of four words will be.



This figure is from Chapter 5 of Stallings: "Cryptography and Network Security", Fourth Edition

The Algorithmic Steps in Going from a 4-Word Round Key to the Next 4-Word Round Key

- Let's say that we have the four words of the round key:

$$w_i \; w_{i+1} \; w_{i+2} \; w_{i+3}$$

  Assuming that i is a multiple of 4, these will serve as the round key for the $(i/4)^{th}$ round. For example, w_4, w_5, w_6, w_7 is the round key for round 1, the sequence of words w_8, w_9, w_{10}, w_{11} the round key for round 2, and so on.

- Now we need to figure out the words

$$w_{i+4} \quad w_{i+5} \quad w_{i+6} \quad w_{i+7}$$

  from $w_i \quad w_{i+1} \quad w_{i+2} \quad w_{i+3}$.

- From the figure on the previous slide, we write

$$\begin{aligned} w_{i+5} &= w_{i+4} \otimes w_{i+1} && (1) \\ w_{i+6} &= w_{i+5} \otimes w_{i+2} && (2) \\ w_{i+7} &= w_{i+6} \otimes w_{i+3} && (3) \end{aligned}$$

  Note that except for the first word in a new 4-word grouping, each word is an XOR of the previous word and the corresponding word in the previous 4-word grouping.

- So now we only need to figure out w_{i+4}. This is the beginning word of each 4-word grouping in the key expansion. The beginning word of each round key is obtained by:

$$w_{i+4} \quad = \quad w_i \otimes g(w_{i+3}) \tag{4}$$

  That is, the first word of the new 4-word grouping is to be obtained by XOR'ing the first word of the last grouping with what is returned by applying a function $g()$ to the last word of the previous 4-word grouping.

- The function $g()$ consists of the following three steps:
  - Perform a one-byte left circular rotation on the argument 4-byte word.
  - Perform a byte substitution for each byte of the word returned by the previous step by using the same 16×16 lookup table as used in the **SubBytes** step of the round.
  - XOR the bytes obtained from the previous step with what is known as a **round constant**. The **round constant** is a word whose three rightmost bytes are always zero. Therefore, XOR'ing with the round constant amounts to XOR'ing with just its leftmost byte.

- The **round constant** for the i^{th} round is denoted $Rcon[i]$. Since, by specification, the three righmost bytes of the round

constant are zero, we can write it as shown below. The left hand side of the equation below stands for the round constant to be used in the i^{th} round. The right hand side of the equation says that the rightmost three bytes of the round constant are zero.

$$Rcon[i] \quad = \quad (RC[i], 0, 0, 0)$$

- The only non-zero byte in the round constants, $RC[i]$, obeys the following recursion:

$$\begin{aligned} RC[1] &= 1 \\ RC[j] &= 2 \times RC[j-1] \end{aligned}$$

An Example of Key Expansion

- To illustrate how the encryption key is expanded into a key schedule consisting of 44 4-byte words, let's consider the following example from Stallings: "Cryptography and Network Security". Let's assume that the encryption key is

$$EA\ D2\ 73\ 21\ B5\ 8D\ BA\ D2\ 31\ 2B\ F5\ 60\ 7F\ 8D\ 29\ 2F$$

- We first represent this in form of a 4×4 key array:

$$\begin{bmatrix} EA & B5 & 31 & 7F \\ D2 & 8D & 2B & 8D \\ 73 & BA & F5 & 29 \\ 21 & D2 & 60 & 2F \end{bmatrix}$$

$$\Downarrow$$

$$[\ w_0, \quad w_1, \quad w_2, \quad w_3\ ]$$

  The bytes are arranged column-wise. That is, the first four bytes of the key are placed in the first column, and so on.

- So the columns of the encryption key arranged in the form of a matrix as shown above yield the first four words, w_0, w_1, w_2, w_3,

of the 44-word key schedule. The plaintext is bitwise XOR'ed with the first four words of the key schedule. Of course, at this point these four words from the key schedule are the same as the encryption key.

- To derive the next four words, w_4, w_5, w_6, w_7, of the key schedule, we need to first derive w_4 from w_0 and w_3 by (see Equation (4) on slide 22):

$$\begin{aligned} w_4 &= w_0 \otimes g(w_3) \\ &= (EA\ D2\ 73\ 21) \otimes g(7F\ 8D\ 29\ 2F) \end{aligned} \quad (5)$$

- The first step of the $g()$ function calls for a one-byte left circular rotation. That gives is the word $8D\ 29\ 2F\ 7F$.

- The second step of $g()$ calls for byte-by-byte substitution for each byte in $8D\ 29\ 2F\ 7F$ using the same S-box that is used in the substitution step of each round. We get the following substitutions from the S-Box:

```
8D   =>   5D
29   =>   A5
2F   =>   15
7F   =>   D2
```

Therefore the output of the second step of $g()$ is $5D\ A5\ 15\ D2$.

- The final step of the $g()$ function consists of XOR'ing the output of the previous step with the **round constant** $Rcon[1]$, which is given by the byte pattern 01 00 00 00. XOR'ing $5D\ A5\ 15\ D2$ with 01 00 00 00 gives us $5C\ A5\ 15\ D2$.

- We now go back to Equation (5) on the previous slide and plug the result obtained above — $5C\ A5\ 15\ D2$ —- for $g()$ in that equation. Evaluating the XOR in that equation gives us for the first word of round key for the first round:

$$\begin{aligned} w_4 &= (EA\ D2\ 73\ 21) \otimes (5C\ A5\ 15\ D2) \\ &= (11101010\ 11010010\ 01110011\ 00100001) \otimes \\ &\quad (01011100\ 10100101\ 00010101\ 11010010) \\ &= (10110110\ 01110111\ 01100110\ 11110011) \\ &= B6\ 77\ 66\ F3 \end{aligned}$$

- Now that we have the first word of the 4-word round key $[w_4, w_5, w_6, w_7]$ needed by the first round, we can obtain the other words by using Equations (1), (2), and (3) on slide 21:

$$\begin{aligned} w_5 &= w_4 \otimes w_1 \\ &= (B6\ 77\ 66\ F3) \otimes (B5\ 8D\ BA\ D2) \\ &= (03\ FA\ DC\ 21) \end{aligned}$$

$$
\begin{aligned}
w_6 &= w_5 \otimes w_2 \\
&= (03\ FA\ DC\ 21) \otimes (31\ 2B\ F5\ 60) \\
&= (32\ D1\ 29\ 41)
\end{aligned}
$$

$$
\begin{aligned}
w_7 &= w_6 \otimes w_3 \\
&= (32\ D1\ 29\ 41) \otimes (7F\ 8D\ 29\ 2F) \\
&= (4D\ 5C\ 00\ 6E)
\end{aligned}
$$

- So now we have the full round key — $[w_4, w_5, w_6, w_7]$ — for the first round.

- We would need to iterate this process to generate all the words, w_0 through w_{43}, of the key schedule.