**4. Advanced Encryption Standard**

Al-Mustanseriya University Class: Third Year

Engineering College Subject: Ciphering & Data Security

Computer and Software Engineering Dep. Lecture -7-

**4.1 The Origins of AES**

The principal drawback of DES & 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid-1970s hardware implementation and does not produce efficient software code. 3DES, which has three times as many rounds as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64-bit block size. For reasons of both efficiency and security, a larger block size is desirable.

Because of these drawbacks, 3DES is not a reasonable candidate for long-term use. As a replacement, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have a security strength equal to or better than 3DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits.

In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. NIST completed its evaluation process and published a final standard in November of 2001. NIST selected Rijndael as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen.

**4.2 The AES Cipher**

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters depend on the key length (Table 4.1). In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

Table 4.1: AES Parameter

|  |  |  |  |
| --- | --- | --- | --- |
| Key size (words/bytes/bits) | 4/16/128 | 6/24/192 | 8/32/256 |
| Plaintext block size (words/bytes/bits) | 4/16/128 | 4/16/128 | 4/16/128 |
| Number of rounds | 10 | 12 | 14 |
| Round key size (words/bytes/bits) | 4/16/128 | 4/16/128 | 4/16/128 |
| Expanded key size (words/bytes) | 44/176 | 52/208 | 60/240 |

Rijndael was designed to have the following characteristics:

* Resistance against all known attacks
* Speed and code compactness on a wide range of platforms
* Design simplicity

Figure (4.1) shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. These operations are depicted in figure (4.2a). Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key (figure 4.2b).

Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the w matrix.

Before delving into details, we can make several comments about the overall AES structure:

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that in the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. Two of the AES finalists, including Rijndael, do not use a Feistel structure but process the entire data block in parallel during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round; these are indicated in figure (4.1).
3. Four different stages are used, one of permutation and three of substitution:
   1. Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
   2. ShiftRows: A simple permutation
   3. MixColumns: A substitution that makes use of arithmetic over GF(28)
   4. AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key
4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure (4.3) depicts the structure of a full encryption round.

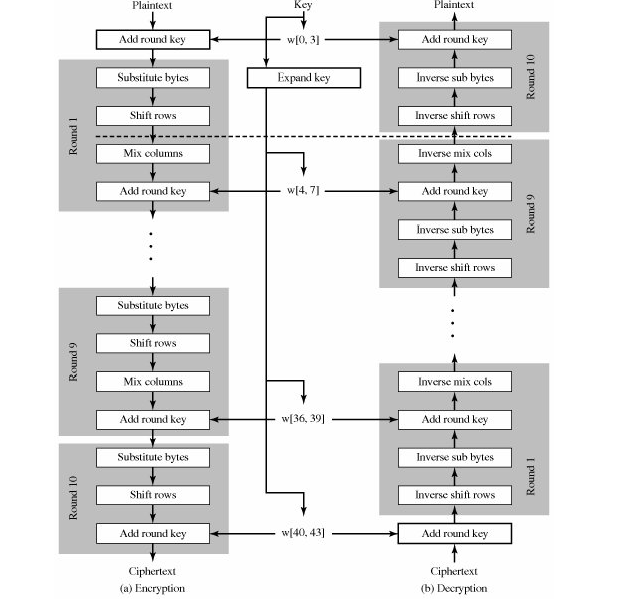


Figure 4.1: AES Encryption and Decryption

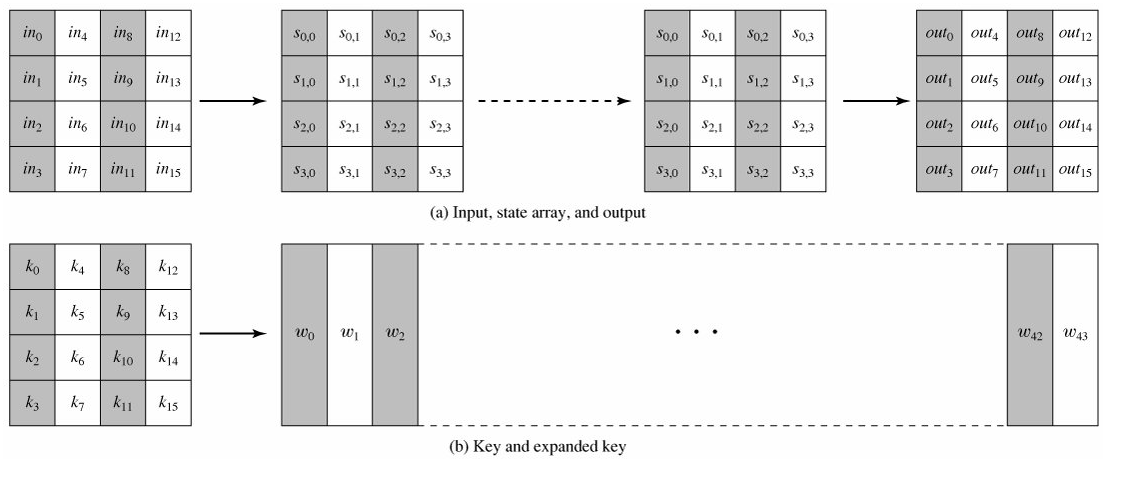


Figure 4.2: AES Data Structures

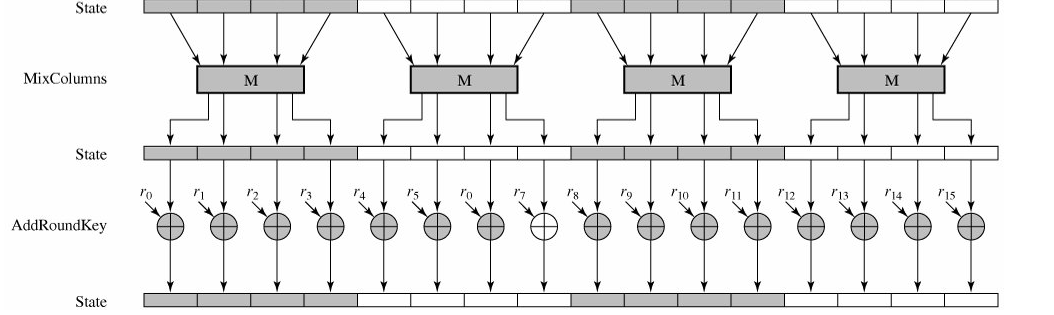
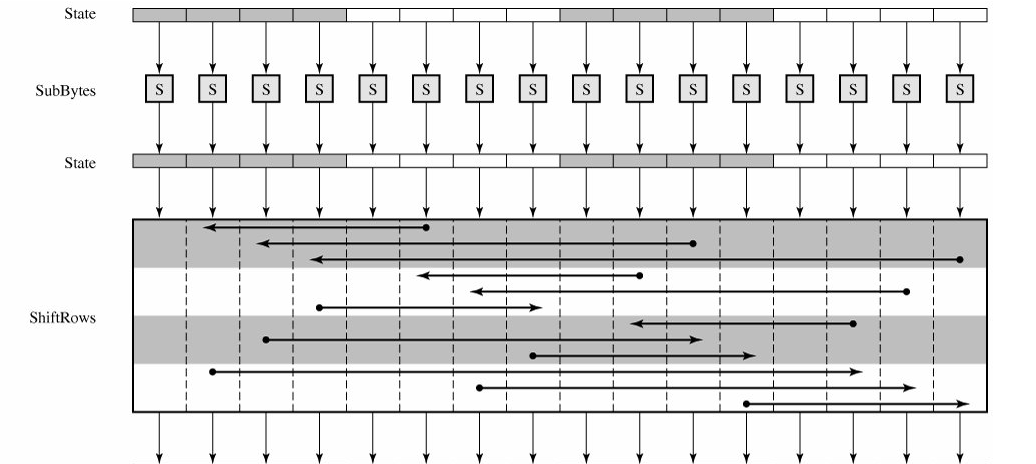


Figure 4.3: AES Encryption Round

1. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
2. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.
3. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that A A B=B.
4. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.
5. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure (4.1) lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.
6. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

AES uses arithmetic in the finite field GF(28), with the irreducible polynomial

m(x) = x8 + x4 + x3 + x + 1.

**4.3 Substitute Bytes Transformation**

The forward substitute byte transformation, called SubBytes, is a simple table lookup (Figure 4.4a). AES defines a 16 x 16 matrix of byte values, called an S-box (Table 4.2a), that contains a permutation of all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

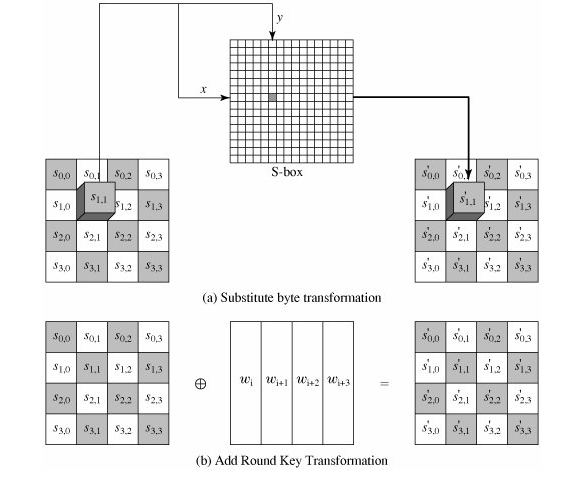
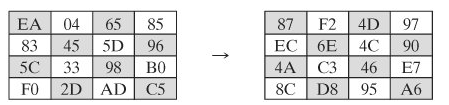


Figure 4.4: AES Byte-Level Operations

Here is an example of the SubBytes transformation:



The S-box is constructed in the following fashion:

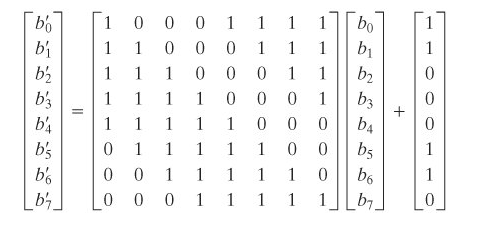
1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02},.... {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row x, column y is {xy}.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(28); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled (b7, b6, b5, b4, b3, b2, b1, b0). Apply the following transformation to each bit of each byte in the S-box:

Equation 5-1:



where ci is the ith bit of byte c with the value {63}; that is, (c7c6c5c4c3c2c1c0) = (01100011). The prime (') indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows:

Equation 5-2:



Equation 5-2 has to be interpreted carefully. In ordinary matrix multiplication, each element in the product matrix is the sum of products of the elements or one row and one column. In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Further, the final addition shown in equation 5-2 is a bitwise XOR.

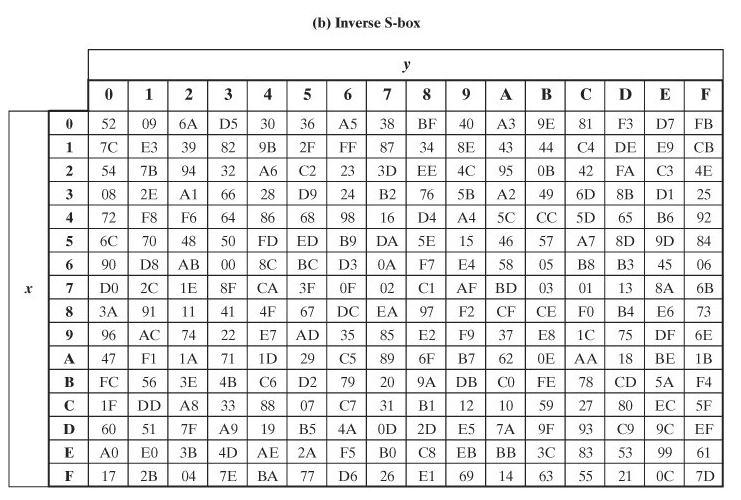
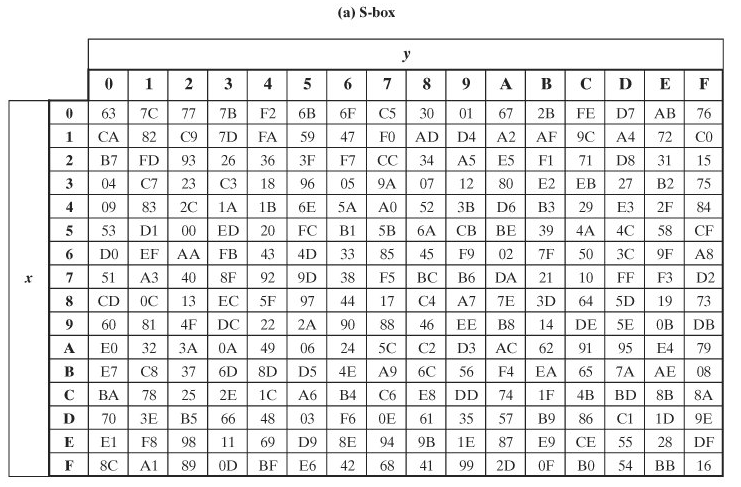
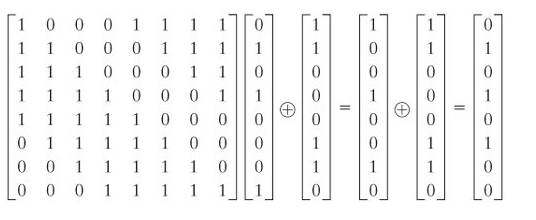


Table 4.2: AES S-Boxes

As an example, consider the input value {95}. The multiplicative inverse in GF(28) is {95}1 = {8A}, which is 10001010 in binary. Using equation 5-2:



The result is {2A}, which should appear in row {09} column {05} of the S-box. This is verified by checking table (4.2a).

The inverse substitute byte transformation, called InvSubBytes, makes use of the inverse S-box shown in table 4.2b. Note, for example, that the input {2A} produces the output {95} and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation in equation 4.1 followed by taking the multiplicative inverse in GF(28). The inverse transformation is:



where byte d = {05}, or 00000101. We can depict this transformation as follows:

