[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy — Winter 2023-2024

Scribed by: Manas Jitendrakumar Ingle (202151086) — Lecture 1,2 (Week 5)

1 Advanced Encryption Standard:

- It is Standardized by NIST.
- Rijndael - winner of Advanced Encryption Standard Competition.
- Winner of the Competition was named AES.

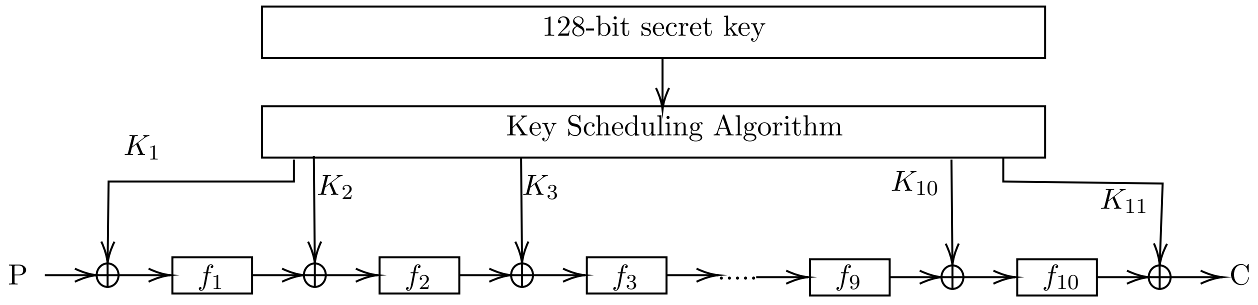
AES is based on -

1. Iterative block cipher.
2. It is based on SPN.

Types of AES:

1. **AES - 128**
   (a) Block size = 128 bit
   (b) Number of Rounds = 10
   (c) Secret key size = 128 bit
2. **AES - 192**
   (a) Block size = 128 bit
   (b) Number of Rounds = 12
   (c) Secret key size = 192 bit
3. **AES - 256**
   (a) Block size = 128 bit
   (b) Number of Rounds = 14
   (c) Secret key size = 256 bit

2 AES 128



The diagram illustrates the process of encrypting plaintext using the AES 128 algorithm. The encryption begins with a 128-bit secret key, which undergoes a Key Scheduling Algorithm to generate 11 round keys, each also 128 bits in length.

The plaintext, denoted as P, is then XOR-ed with the first round key, denoted as K_1. This intermediate result is fed into the first round function, f_1, which produces a 128-bit output. This output is further processed by XOR-ing with the second round key, K_2, and then passed through the second round function, f_2. This process iterates through f_3 to f_{10}, with each function generating a new output and XOR-ing it with the corresponding round key.

Finally, the output of the tenth round function, f_{10}, undergoes XOR with the eleventh round key, K_{11}, to produce the ciphertext, denoted as C.

Given the ciphertext, decryption process will begin with xoring the ciphertext with K_{11}(last round key) and this output will be input to inverse of f_{10} function. Inverse of f_{10} function will generate a 128-bit output which will be xored with K_{10}, and output will be passed to inverse of f_9 function, and so on it will be continued till inverse of f_1 function. The output will be xored with K_1(first round key) to get the plaintext.

Note: The decryption process described above clearly states that the round functions $f_1, f_2, ..., f_{10}$ must be invertible. Hence, this structure is different from Feistel Network where the invertibility of round functions does not matter.
Now, we need to understand the following things:

- Round Functions
- Key Scheduling Algorithm

Round Functions of AES-128:

$f_1, f_2, \ldots, f_{10}$

1. $f_1 = f_2 = f_3 \cdots = f_9$
2. f_{10} is different from $f_i, i = 1, 2, \ldots, 9$

First 9 round functions are exactly same and the 10th round function is different from other 9 round functions.

- The first 9 round functions $(i.e, f_1, f_2, \ldots, f_9)$ are based on the following functions:

  1. Sub-bytes
  2. Shift rows
  3. Mix columns

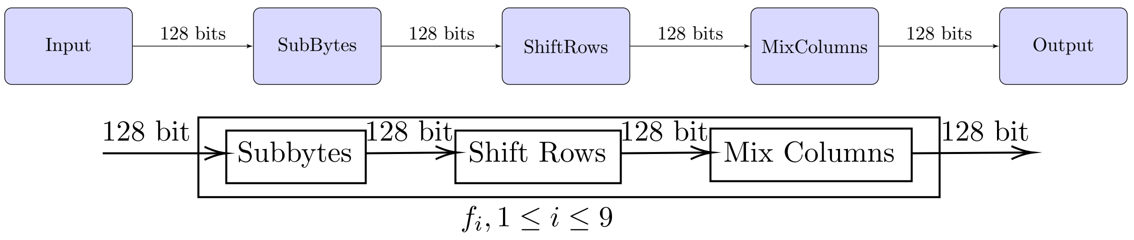
  $$f_i : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$
  $$Mixcol(Shiftrows(sub-bytes(x))) = f_i(x)$$

  The 10th round function $(i.e, f_{10})$ is based on:

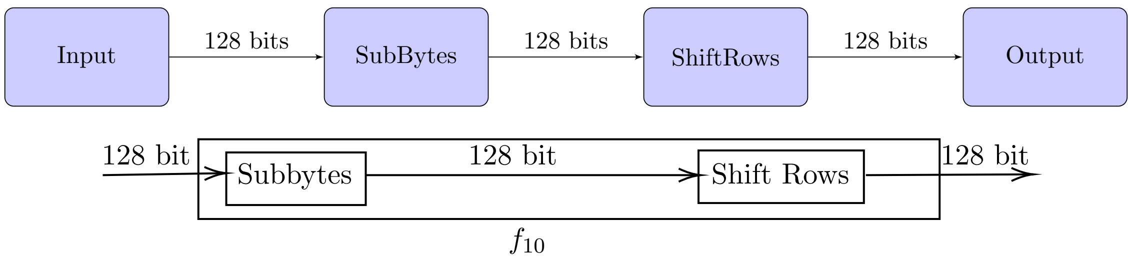
  1. Sub-bytes
  2. Shift rows

  $$f_{10} : \{0,1\}^{128} \rightarrow \{0,1\}^{128}$$

  The only difference between the last round function and the rest is that the last round function do not make use of mix columns.



Block Diagram for AES-128 for first 9 rounds



For the Last Round function f_{10}

Subbytes

Subbytes is a bijective mapping from 128-bit to 128-bit.

Let $X = x_0 x_1 \ldots x_{15}$ where the size of each x_i is 8 bits.

X $\xrightarrow{Mapping}$ S

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} \to \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix}$$

S-Box Definition:

$S : \{0,1\}^8 \to \{0,1\}^8$

S-Box Initialization:

$S(0) = 0$

Hexadecimal Representation of Constants:

C = $C_7C_6C_5C_4C_3C_2C_1C_0 \leftarrow (01100011) = (63)_{16}$

S-Box Transformation:

For an input s_{ij} represented as $(a_7a_6a_5a_4a_3a_2a_1a_0)$:

$$b_i = (a_i + a_{i+4} \mod 8 + a_{i+5} \mod 8 + a_{i+6} \mod 8 + a_{i+7} \mod 8 + C_i) \mod 2$$

Output:

$$s'_{ij} = (b_7b_6b_5b_4b_3b_2b_1b_0)$$

Hence, S'_{ij} is computed for each S_{ij} and the output matrix is the output of subbyte function.

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \xrightarrow{Subbyte} \begin{bmatrix} S'_{00} & S'_{01} & S'_{02} & S'_{03} \\ S'_{10} & S'_{11} & S'_{12} & S'_{13} \\ S'_{20} & S'_{21} & S'_{22} & S'_{23} \\ S'_{30} & S'_{31} & S'_{32} & S'_{33} \end{bmatrix}$$

2.0.1 Mix Columns

Mix columns, again, is a mapping from 128-bit to 128-bit. It also takes a 4×4 matrix as input (the output of Shift Rows function).

$$\text{Mix Columns: } \{0,1\}^{128} \to \{0,1\}^{128}$$
$$(S_{ij})_{4\times 4} \xrightarrow{MixColumns} (S^{'}_{ij})_{4\times 4}$$

Consider the column $c \in 0, 1, 2, 3$ of matrix S.

$$\text{column} = \begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix}$$

The Mix Columns function is defined as follows. For i = 0 to i = 3, let t_i be the polynomial constructed from S_{ic}. Define four polynomials as:

$$u_0 = [(x * t_0) + (x+1) * t_1 + t_2 + t_3] \text{ mod } (x^8 + x^4 + x^3 + x + 1)$$
$$u_1 = [t_0 + (x * t_1) + (x+1) * t_2 + t_3] \text{ mod } (x^8 + x^4 + x^3 + x + 1)$$
$$u_2 = [t_0 + t_1 + (x * t_2) + (x+1) * t_3] \text{ mod } (x^8 + x^4 + x^3 + x + 1)$$
$$u_3 = [(x+1) * t_0 + t_1 + t_2 + (x * t_3)] \text{ mod } (x^8 + x^4 + x^3 + x + 1)$$

Now, $S^{'}_{ij}$ is the binary 8-bits constructed using u_i. Therefore,

$$\begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix} \xrightarrow{MixColumns} \begin{bmatrix} S^{'}_{0c} \\ S^{'}_{1c} \\ S^{'}_{2c} \\ S^{'}_{3c} \end{bmatrix}$$

Applying Mix Columns to each columns, will give us the entire $(S^{'}_{ij})_{4\times 4}$ matrix. Therefore, Mix Column can be defined as a matrix multiplication as:

$$(S^{'}_{ij})_{4\times 4} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \times \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \text{ mod } (x^8 + x^4 + x^3 + x + 1)$$

In terms of hexadecimal, the above multiplication can be represented as:

$$(S^{'}_{ij})_{4\times 4} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \text{ mod } (x^8 + x^4 + x^3 + x + 1)$$

The polynomial $(x^8 + x^4 + x^3 + x + 1)$ is a primitive polynomial, hence, it is possible to construct the inverse of the Mix Columns function.

Lecture 2 [16/02/24]

Find $\begin{bmatrix} S'_{00} \\ S'_{10} \\ S'_{20} \\ S'_{30} \end{bmatrix}$ after doing the Mix Column operation on $\begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix}$ where $S_{00} = 95, S_{10} = 65, S_{20} = fd, S_{30} = f3$.

Solution:

$$\begin{gathered} S_{00} = 95 = 10010101 = x^7 + x^4 + x^2 + 1 \\ S_{10} = 65 = 01100101 = x^6 + x^5 + x^2 + 1 \\ S_{20} = fd = 11111101 = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1 \\ S_{30} = f3 = 11110011 = x^7 + x^6 + x^5 + x^4 + x + 1 \end{gathered}$$

Now, let's calculate S'_{00}.

$S'_{00} = (x * S_{00} + (x+1) * S_{10} + S_{20} + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{00} = ((x^8 + x^5 + x^3 + x) + (x^7 + x^5 + x^3 + x^2 + x + 1) + S_{20} + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{00} = (x^8 + x^7 + x^3 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{00} = (x^7 + x^4) = 10010000 = (90)_{16}$

Similarly we can compute S'_{10}, S'_{20} and S'_{30}. Let's calculate each one of them.

$S'_{10} = (S_{00} + x * S_{10} + (x+1) * S_{20} + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{10} = (S_{00} + (x^7 + x^6 + x^3 + x) + (x^8 + x^2 + x + 1) + S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{10} = (x^8 + x^7 + x^5 + x^3 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{10} = (x^7 + x^5 + x^4) = 10110000 = (b0)_{16}$

Similarly,

$S'_{20} = (S_{00} + S_{10} + x * S_{20} + (x+1) * S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{20} = (S_{00} + S_{10} + (x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x) + (x^8 + x^4 + x^2 + 1)) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{20} = (x^4 + x^3 + x^2 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S'_{20} = (x^4 + x^3 + x^2 + x + 1) = 00011111 = (1f)_{16}$

Similarly,

$S_{30}^{'} = ((x+1) * S_{00} + S_{10} + S_{20} + x * S_{30}) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{30}^{'} = ((x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) + S_{10} + S_{20} + (x^8 + x^7 + x^6 + x^5 + x^2 + x)) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{30}^{'} = (x^7 + x^6 + 1) \bmod (x^8 + x^4 + x^3 + x + 1)$

$S_{30}^{'} = (x^7 + x^6 + 1) = 11000001 = (c1)_{16}$

Therefore, we have,

$$\begin{bmatrix} S_{00}^{'} \\ S_{10}^{'} \\ S_{20}^{'} \\ S_{30}^{'} \end{bmatrix} = \begin{bmatrix} 90 \\ b0 \\ 1f \\ c1 \end{bmatrix}$$

.

Key Scheduling Algorithm of AES 128

The key scheduling algorithm of AES-128 takes 128-bit key as input and generate 11 round keys of 128-bits each.

$$\text{key} = (key[0], key[1], ..., key[15]) \text{ where each } key[i] \text{ is 1 byte long}$$

We will generate 44 words (32-bit) - w[0], w[1],...., w[43]. We can see that (32 * 44)/128 = 11. Therefore, we can generate 11 round keys from these 44 words.

There are two functions involved in key scheduling algorithm. These are given below:

1. ROTWORD($B_0B_1B_2B_3$): It takes a word as input and performs left circular shift of its bytes once (or left circular shift by 8 bits). Here, B_0, B_1, B_2, B_3 are bytes of the input word. The output of the ROTWORD function is:

$$\text{ROTWORD}(B_0B_1B_2B_3) = B_1B_2B_3B_0$$

2. SUBWORD($B_0B_1B_2B_3$): It takes a word as input and performs the Subbyte function (defined in round function of AES) of its bytes B_0, B_1, B_2, B_3. The output of SUBWORD function is:

$$\text{SUBWORD}(B_0B_1B_2B_3) = B_0^{'}B_1^{'}B_2^{'}B_3^{'}$$
$$\text{where } B_i^{'} = Subbytes(B_i) \ \forall \ i \in \{0, 1, 2, 3\}$$

Also, there are ten round constants (which are words, i.e, 32-bit) defined as:

$$RCON[1] = 01000000$$
$$RCON[2] = 02000000$$
$$RCON[3] = 04000000$$
$$RCON[4] = 08000000$$
$$RCON[5] = 10000000$$
$$RCON[6] = 20000000$$
$$RCON[7] = 40000000$$

$$RCON[8] = 80000000$$
$$RCON[9] = 1b000000$$
$$RCON[10] = 36000000$$

Note that the values of constants written above are in hexadecimal. The 44 words are generated using the below algorithm.

For $i = 0$ to 3:

$$w[i] = (Key[4i], Key[4i+1], Key[4i+2], Key[4i+3])$$

For $i = 4$ to 43:

$$\text{temp} = w[i-1]$$

if $i \equiv 0 \mod 4$:

$$\text{temp} = \text{SUBWORD}(\text{ROTWORD}(\text{temp})) \oplus \text{Rcon}[i/4]$$

$$w[i] = w[i-4] \oplus \text{temp}$$

Now, the round keys will be given as:

$$K_1 = w[0]||w[1]||w[2]||w[3]$$
$$K_2 = w[4]||w[5]||w[6]||w[7]$$
$$.$$
$$.$$
$$K_{11} = w[40]||w[41]||w[42]||w[43]$$

Mode of Operation

We know that we can encrypt 128 bit blocks using AES at once. Suppose we want to encrypt more data, let's say 256 bit data. We need some mechanism to encrypt this data. There are certain mode of operations which we will be discussing in encrypting more data than the block length. Few examples of mode of operation are mentioned below.

1. Electronic CodeBook Mode (EBC)
2. Cipher FeedBack Mode (CFB)
3. Cipher Block Chaining Mode (CBC)
4. Output FeedBack Mode (OFB)

Electronic CodeBook Mode

As discussed above, in this mode, the plaintext is divided into continous blocks of l-bit size and each block is encrypted separately. The corresponding ciphertexts are concatenated in the same order to get the final ciphertext.

$$M = m_0||m_1||.....||m_t \text{ (plaintext)}$$
$$len(m_i) = l - bit \text{ (each } m_i \text{ is a block, for AES, l = 128)}$$

Encryption:

$$C = C_0||C_1||.....||C_t$$
$$C_i = Enc(m_i, K) \ \forall \ i \in \{0, 1, ..., t\}$$

Decryption:

$$M = m_0||m_1||.....||m_t$$
$$m_i = Dec(C_i, K)\ \forall\ i \in \{0, 1, ..., t\}$$

The advantages of using ECB mode is that multiple blocks can be encrypted in parallel. However, it will reveal information if few blocks are same, i.e., if $m_i = m_j$ then $C_i = C_j$.

Cipher Block Chaining Mode

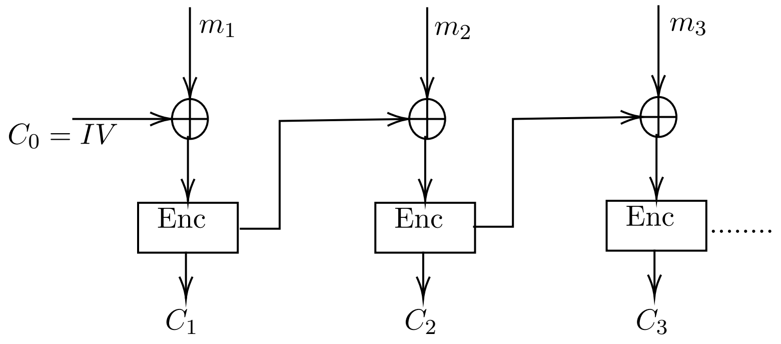
It is the most used mode of operation. It is used in applications like WhatsApp. There is a block of l-bit which is public. It is known as Initialization Vector (IV).

Encryption:

$$M = m_1||m_2||.....||m_t \text{ (plaintext)}$$
$$len(m_i) = l - bit \text{ (each } m_i \text{ is a block, for AES, l = 128)}$$

The encrypted text in CBC mode contains (n+1) blocks for a plaintext of n blocks. The ciphertext is computed as:

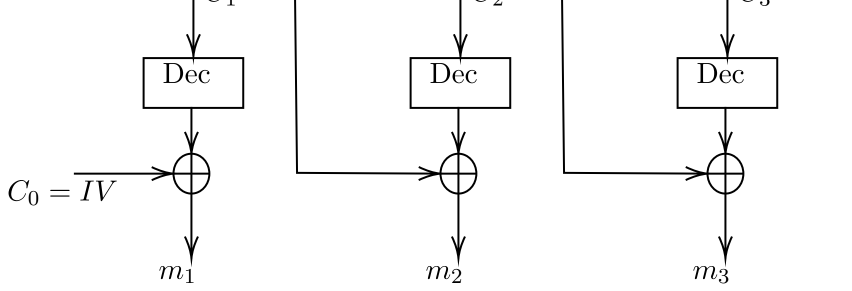
$$C_0 = IV$$
$$C_i = Enc(C_{i-1} \oplus m_i, K)\ \forall\ i \in \{1, 2, 3...., t\}$$
$$C = C_0||C_1||.....||C_t$$



Encryption in CBC Mode

Decryption:

$$m_i = Dec(C_i, K) \oplus C_{i-1}\ \forall\ i \in \{1, 2..., t\}$$
$$\text{where } C_0 = IV$$
$$M = m_1||m_2||.....||m_t$$



Decryption in CBC Mode

Stream Cipher

Stream ciphers encrypt bitwise.

Let $M = m_0 m_1 \ldots m_i$ where $m_i \in \{0, 1\}$ represents the plaintext.

Let $K = k_0 k_1 \ldots k_i$ represent the key.

The ciphertext C is obtained by bitwise XOR (exclusive or) operation:

$$C = M \oplus K = c_0 c_1 \ldots c_i$$

where $c_i = m_i \oplus k_i$.

Shannon's Notion of Perfect Secrecy

Perfect secrecy, as defined by Claude Shannon, ensures that the ciphertext provides no information about the plaintext. This is expressed as:

$$P[M = M_1 | C = CH_1] = P[M = M_1]$$

If this condition holds, the algorithm is considered perfectly secure, meaning the ciphertext reveals nothing about the plaintext.

Conditions for Perfect Secrecy in Stream Cipher

To achieve perfect secrecy in a stream cipher, the following conditions must be satisfied:

1. **Key Length:** The length of the key must be greater than or equal to the length of the message.

2. **Unique Keys:** The same key should not be used to encrypt different messages. Each message requires a unique key, adhering to the concept of One Time Padding.

Perfect secrecy, when implemented in a stream cipher with these conditions, ensures a high level of security by preventing the ciphertext from revealing any information about the plaintext.