# **ASM-Networking**

Our ASM-Networking includes three main components:

1. ASM DLL, an assembly program, is responsible for encrypting and decrypting a message using AES algorithm.
2. Client Side which is responsible for sending an encrypted message and key to server side.
3. Server Side which is responsible for receiving an encrypted message and key.

This project reads a text and an encryption key from the user (using GUI in C#), sends that text and key to your assembly program to encrypt it using AES algorithm then send it to server side. The server side receives the encrypted message and key and sends them to your assembly program to decrypt it. Then, it will show the decrypted text for the end user (using GUI in C#).

**Details**

The **Advanced Encryption Standard (AES),** also known by its original name Rijndael is a specification for the [encryption](https://en.wikipedia.org/wiki/Encryption) of electronic data. AES operates on a 4 × 4 [column-major order](https://en.wikipedia.org/wiki/Column-major_order) array of bytes (128-bit) as a plain text. Encryption consists of 10 rounds of processing for 128-bit keys. The key is a 4 x 4 column major order of bytes. Except for the last round in each case, all other rounds are identical as shown in the figure 1. Each round includes 4 steps that will be discussed in details.

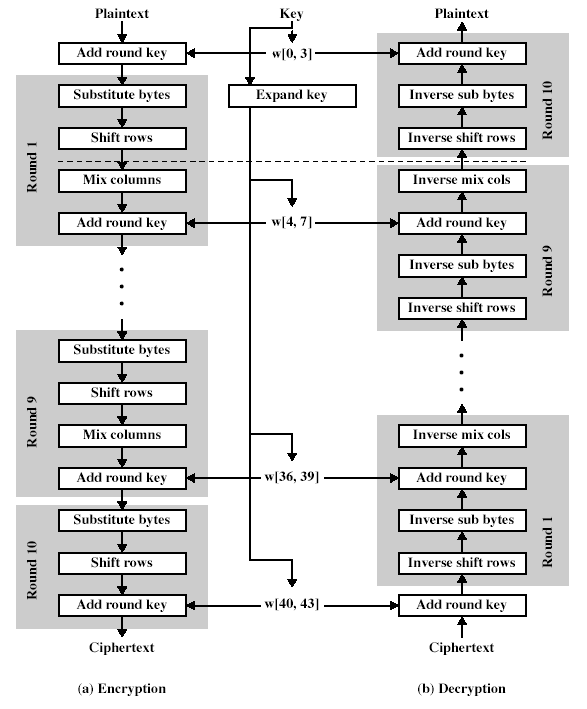


Figure 1. AES Algorithm

**Input: 1 byte array (4 X 4 plain text matrix).**

**1 byte array (4 X 4 key matrix).**

**Output: 1 byte array (4 X 4 cipher text matrix).**

**Add Round Key step:**

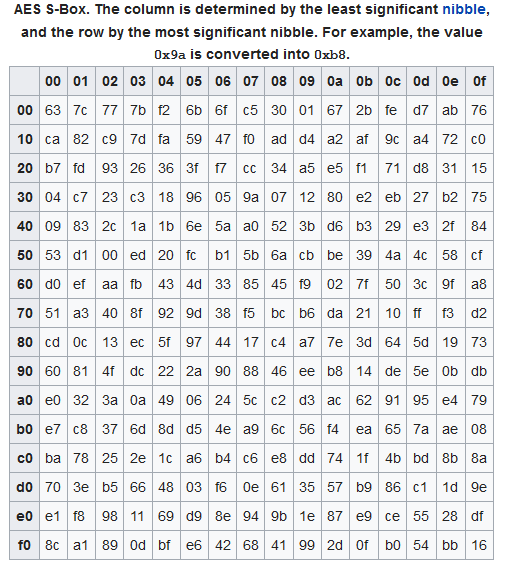
It’s a simple XOR between the plain text and the round key.

It’s the same step in encryption and decryption.

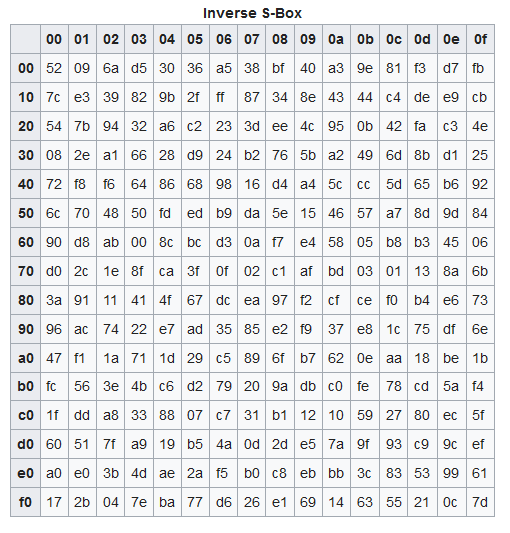
**Substitute Bytes step:**

Each byte is replaced with another according to a [lookup table](https://en.wikipedia.org/wiki/Rijndael_S-box) as follows.

Use the following table in encryption.

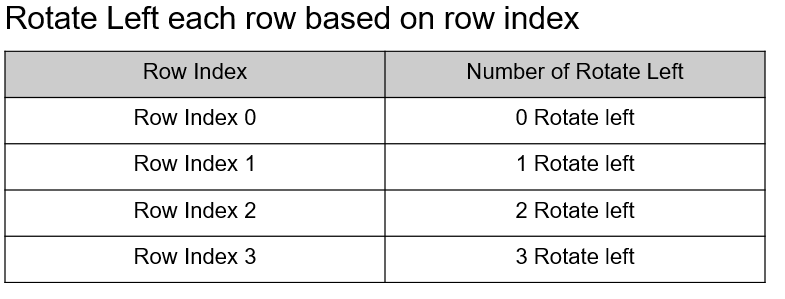


And the following table for decryption.



**Shift Rows Step:**

It’s a transposition step where the last three rows of the state (matrix) are rotated left in encryption and right in decryption a certain number of **bytes** as shown in figure 2.



**Figure 2. Shift rows**

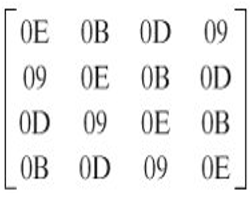
**Mix Columns Step:**

It’s a multiplication step, multiplying the output matrix from shift rows by a fixed matrix using advanced method for multiplication. Use the following matrix in encryption.



**Figure 3. Multiplication matrix for mix columns encryption**

**And use the following matrix in decryption.**



**Figure 4. Multiplication matrix for mix columns decryption**

**Advanced Multiplication:**

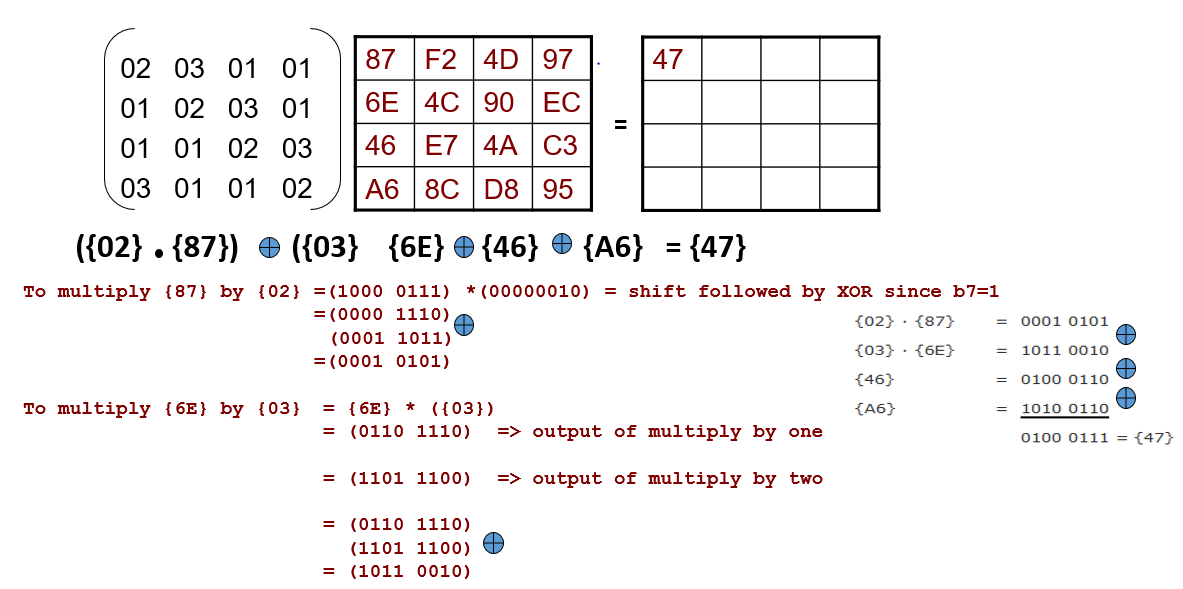
* It depends on **shifting** and **XOR** steps.
* The **addition** is done using **XOR**.
* The details of the steps are described as follows:

**Generally, after shifting the byte to left by one, check the output bit if it’s one then XOR with 0x1B then use this value as the output of multiplication. If it’s zero then use this shifted value as the output of multiplication.**

* Multiply by **01**, means use the same value of byte.
* Multiply by **02**, means shift left one time and check the CF then use the output value as result of multiplication.
* Multiply by **03**, is divided into two multiplication:
  + Multiply by one and keep the value in X0.
  + Take results and multiply it by two. To multiply by two use the above description and keep value in X1.
  + XOR the result of multiplication by one and result of multiplication by two (XOR X0 and X1) and this is the output of multiplication..
* Multiply by **09**, is done as following:
* Multiply by 1 and keep the value in X0.
  + Multiply X0 by 02 and keep value in X1.
  + Multiply X1 by 02 and keep value in X2.
  + Multiply X2 by 02 and keep value in X3.
  + XOR it X0 with X3 and this is the output of multiplication.
* Multiply by **0B**, is done as following:
  + Multiply by 1 and keep the value in X0.
  + Multiply X0 by 02 and keep the value in X1.
  + Multiply X1 by 02 and keep the value in X2.
  + Multiply X2 by 02 and keep the value in X3.
  + XOR ( X3 , X1, X0) and this is the output of multiplication.
* Multiply by **0D**, is done as following:
  + Multiply by 1 and keep the value in X0.
  + Multiply X0 by 02 and keep the value in X1.
  + Multiply X1 by 02 and keep the value in X2.
  + Multiply X2 by 02 and keep the value in X3.
  + XOR (X3, X2, X0) and this is the output of multiplication.
* Multiply by **0E**, is done as following:
  + Multiply by 1 and keep the value in X0.
  + Multiply X0 by 02 and keep the value in X1.
  + Multiply X1 by 02 and keep the value in X2.
  + Multiply X2 by 02 and keep the value in X3.
  + XOR (X3, X2, X1) and this is the output of multiplication.

**Example of Mix Columns:**

Input Matrix



**Figure 5. Example of Mix Columns**

**Generation of Key:**

Given matrix of input key, you’ll generate 10 keys for the 10 rounds. Each Round key depends the previous round key which means key for round 1 depends on the input key and key for round 2 depends on the key of round 1 and so on. The details of generation will be discussed in the following steps:

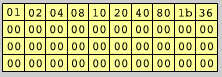
Each column Wi in the new key matrix is output of XORing Wi-1 and W i-4  where Wi-1 isthe previous column in same key matrix, and Wi-4 is the same column in previous key matrix.

Except the first column, as the previous column is in the previous round key. It will pass by a function that contains the following steps:

1. Rotate one byte up to down.
2. Substitute from S-box.
3. XOR with round constant column (choose column based on which round you’re generating the key for).
4. XOR with the same column in the previous key.

****

**Figure 6. Generation of round key**



**Figure 7. Round constant table**

Example to generate W4 as shown in the previous figure:

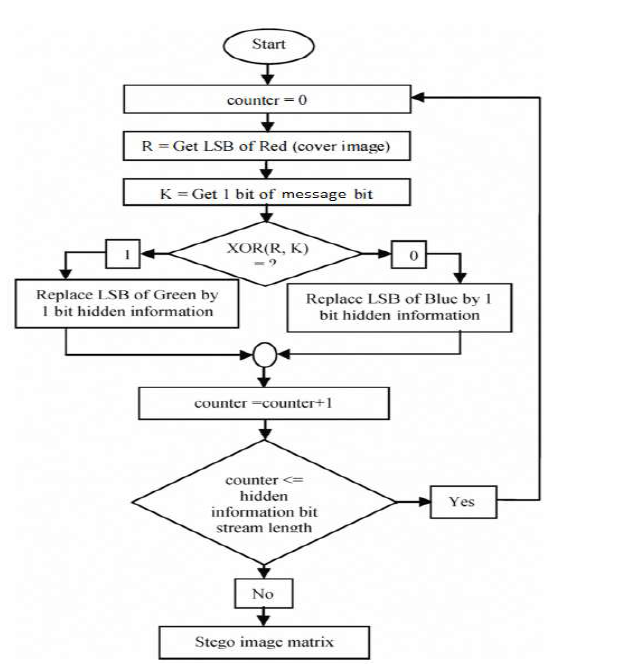
1. Rotate W3 one byte from up to down.
2. Substitute from S-box.
3. XOR with first column in round constant as we generate the key for first round.
4. XOR with W0

**To use the Template:**

1. Call your assembly function instead of the not implemented exception in client and server forms.
2. Run the server side first, by right click on the project, select debug, select run new instance.
3. On server side form, click the start button.
4. Run the client side, by right click on the project, select debug, select run new instance and enter new message and key.
5. The message and key should be 16 character each and click send.
6. Check the output in Server side form.

**Bonus**

* Support Socket programming using Assembly only.
* Implement steganography for embedding the encrypted message in an image and extract an embedded text from image using Assembly only, the image would be send and received through the C# application.



**Figure 8. Flowchart of steganography**

* **Hints**

NASM supports socket programming on Linux.

Substitution in key generation is done using S-box only not Inverse S-box in encryption and decryption.

* **Team members:** 5 members.
* **Responsible TA:** Donia Gamal