Secured Communication

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 of electronic data. AES operates on a 4 × 4 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.

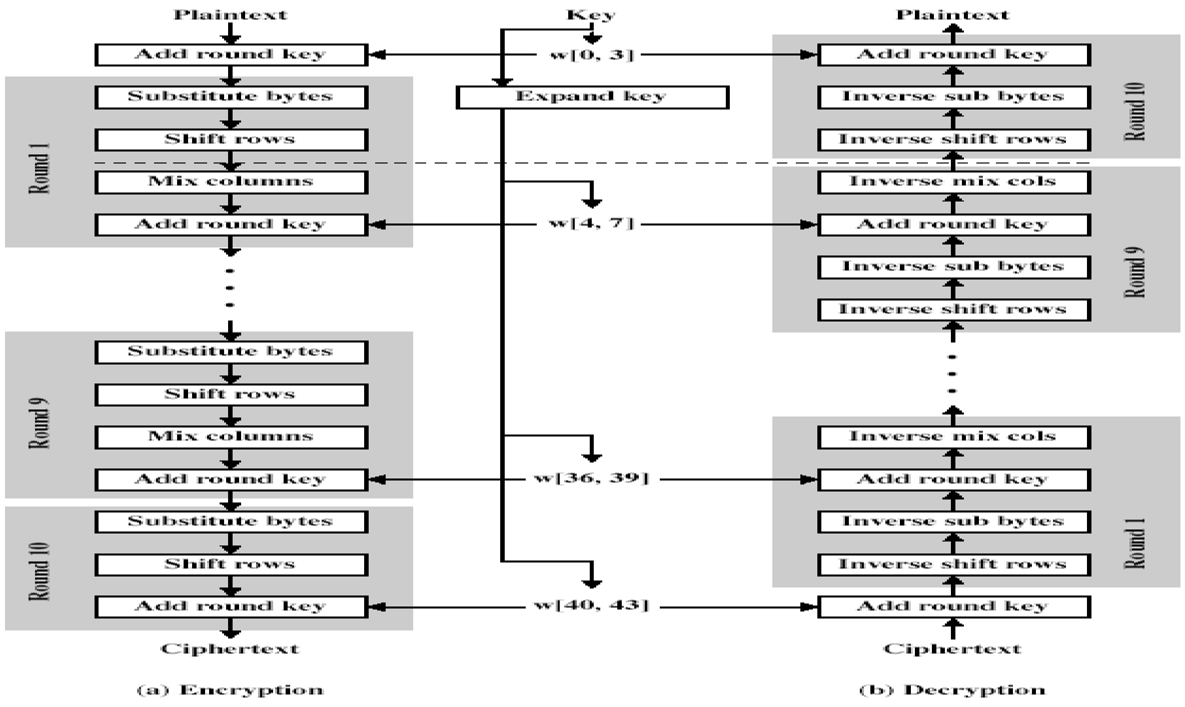


Figure 1. AES Algorithm

Input: 16 byte array (4 X 4 plain text matrix).

16 byte array (4 X 4 key matrix).

Output: 16 byte array (4 X 4 cipher text matrix).

Add Round Key step:

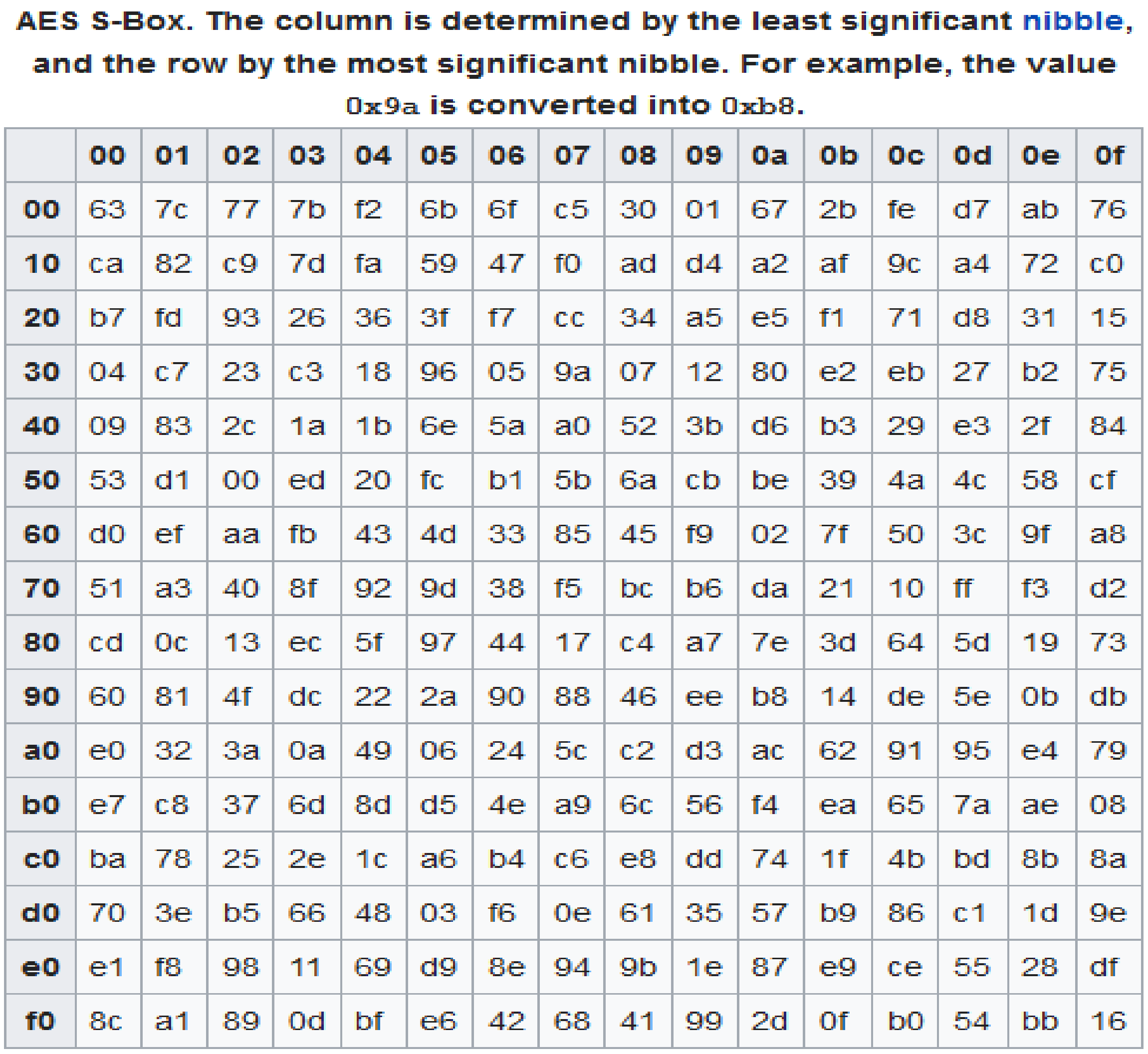
It’s a simple XOR between the plain text and the round key.

It’s the same step in encryption and decryption.

Substitute Bytes step:

Each byte is replaced with another according to a lookup table as follows.

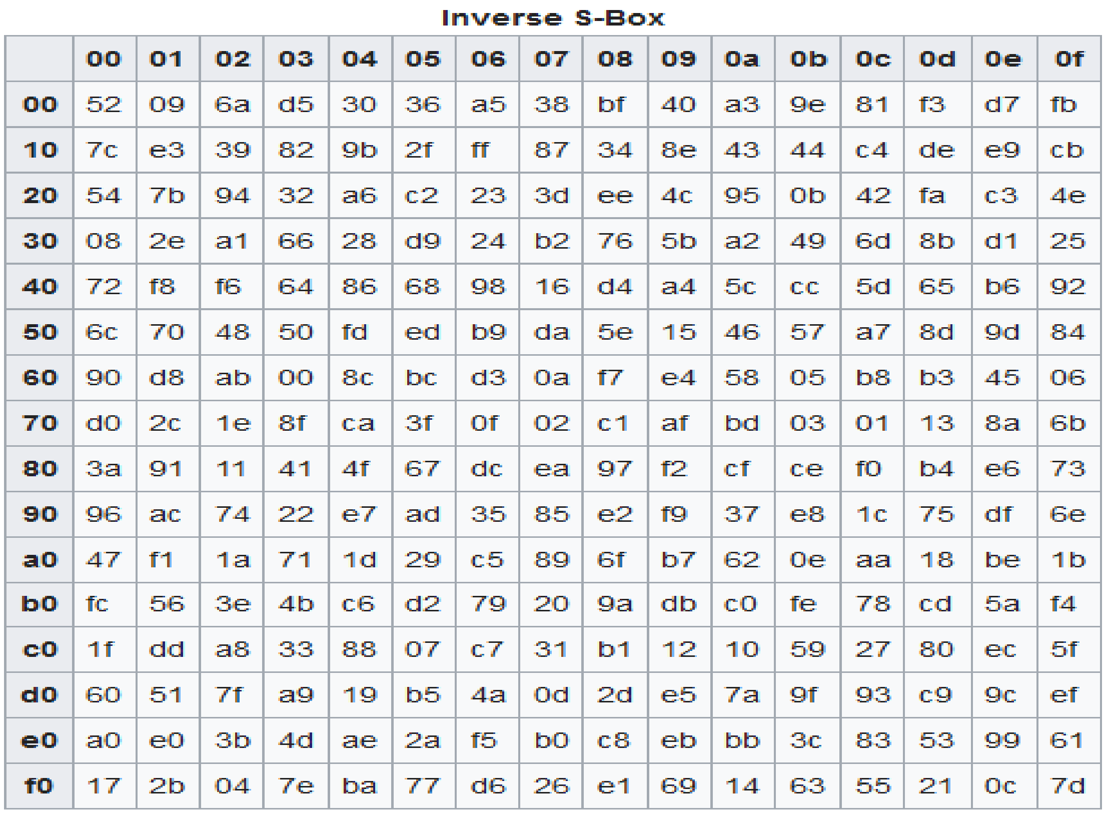
Use the following table in encryption.



AES S-Box. The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value 0x9a is converted into 0xb8.

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

And the following table for decryption.

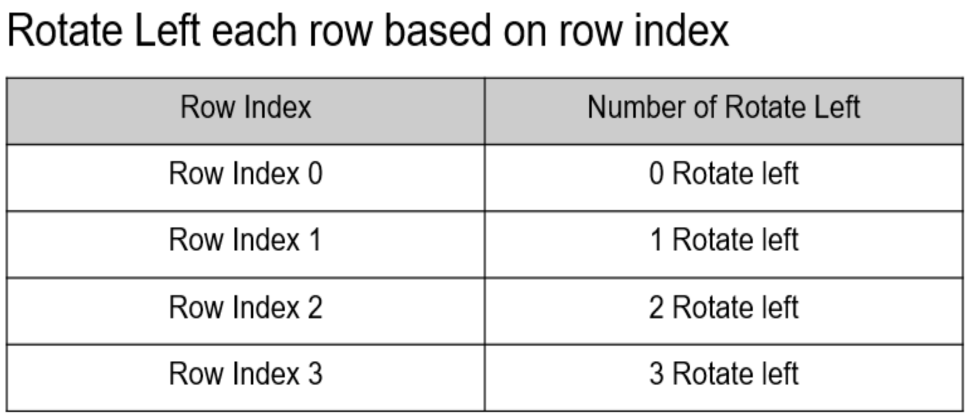


Inverse S-Box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

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.



Rotate Left each row based on row index

Row Index	Number of Rotate Left
Row Index 0	0 Rotate left
Row Index 1	1 Rotate left
Row Index 2	2 Rotate left
Row Index 3	3 Rotate left

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.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Figure 3. Multiplication matrix for mix columns encryption

And use the following matrix in decryption.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

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:

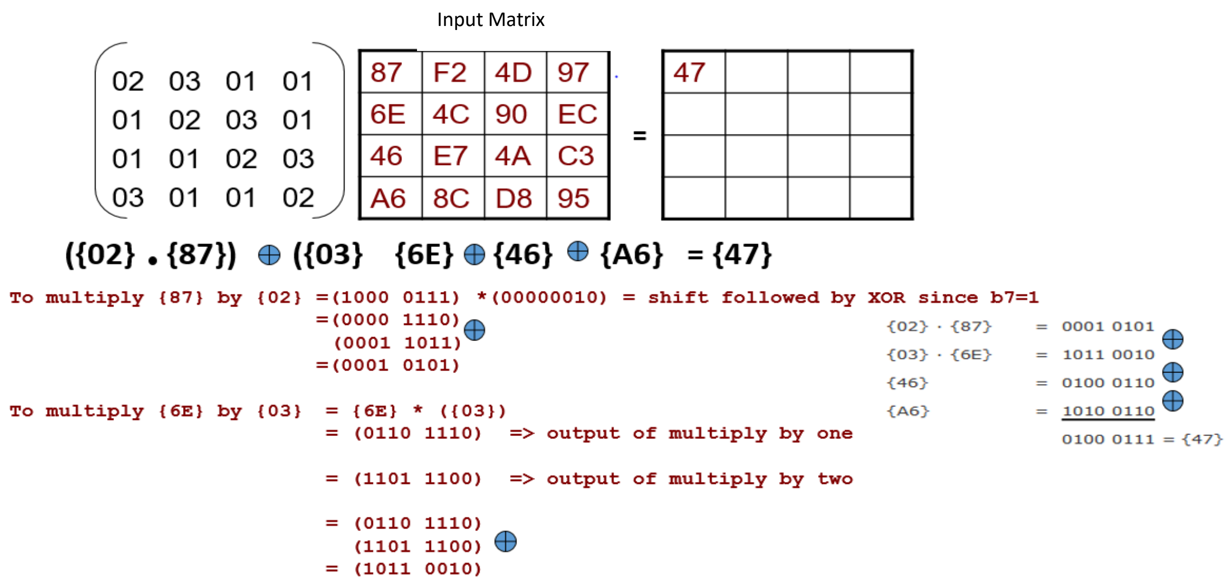

Input Matrix
02 03 01 01
01 02 03 01
01 01 02 03
03 01 01 02
87 F2 4D 97
6E 4C 90 EC
46 E7 4A C3
A6 8C D8 95
=
47
({02} • {87}) ⊕ ({03} {6E} ⊕ {46} ⊕ {A6} = {47}
To multiply {87} by {02} =(1000 0111) *(00000010) = shift followed by XOR since b7=1
=(0000 1110) ⊕
(0001 1011)
=(0001 0101)
To multiply {6E} by {03} = {6E} * ({03})
= (0110 1110) => output of multiply by one
= (1101 1100) => output of multiply by two
= (0110 1110)
(1101 1100) ⊕
= (1011 0010)
{02} · {87} = 0001 0101 ⊕
{03} · {6E} = 1011 0010 ⊕
{46} = 0100 0110 ⊕
{A6} = 1010 0110
0100 0111 = {47}

Figure 5. Example of Mix Columns

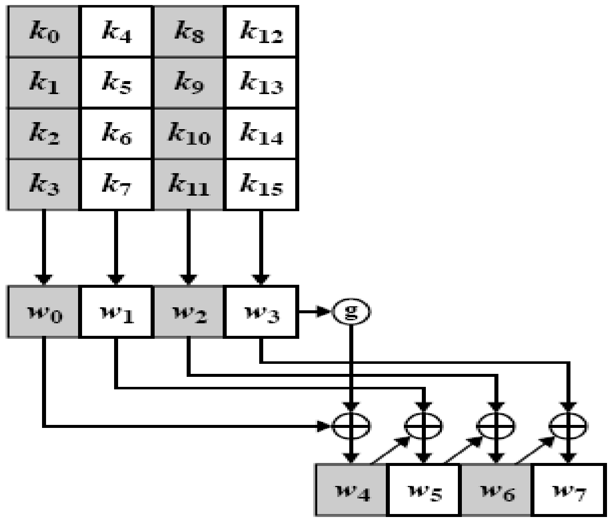
Generation of Key:

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 W_i in the new key matrix is output of XORing W_{i-1} and W_{i-4} where W_{i-1} is the previous column in same key matrix, and W_{i-4} is the same column in previous key matrix.

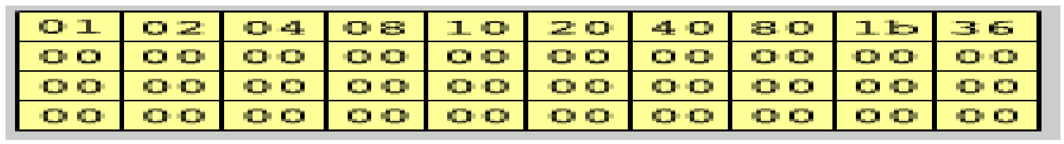
Except the first column, as the previous column is in the previous round key. It will pass by a function that contains the following steps:

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.



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}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7

Figure 6. Generation of round key



01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Figure 7. Round constant table

Example to generate W_4 as shown in the previous figure:

1) Rotate W_3 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 W_0

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.

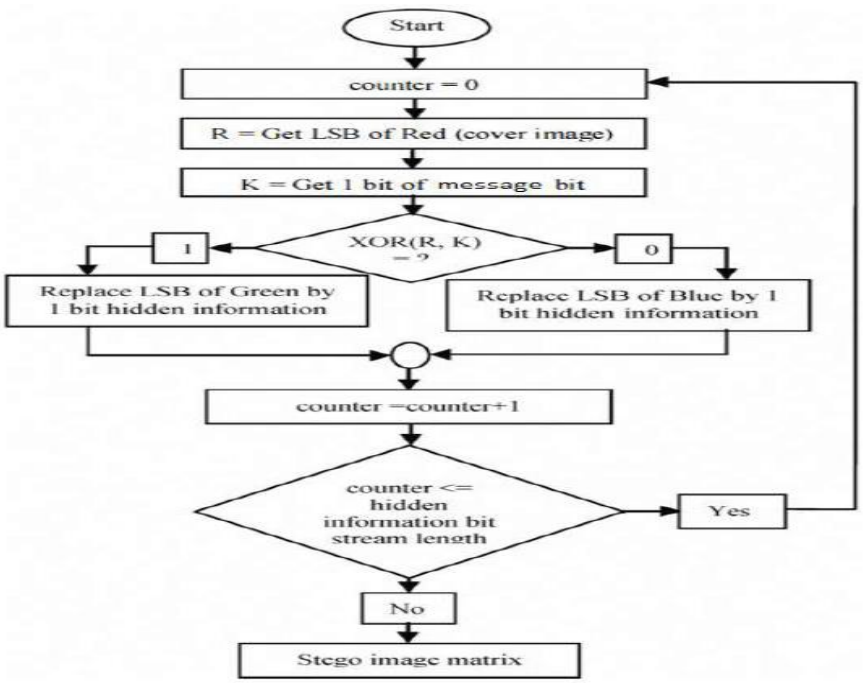

Start
counter = 0
R = Get LSB of Red (cover image)
K = Get 1 bit of message bit
XOR(R, K) = ?
1
0
Replace LSB of Green by 1 bit hidden information
Replace LSB of Blue by 1 bit hidden information
counter =counter+1
counter <= hidden information bit stream length
Yes
No
Stego image matrix

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.