Advanced Encryption Standard (AES)

The whole code is self-explanatory, and comments have been added throughout the code for explanation. Here, I will explain how every part of the code actually works.

1. **Initial Data Chosen**: Random data was chosen for test cases. I created a 2-D array to store 16 bytes of data (total of 128 bits of data).
2. **Initial Key Chosen**: Random key was chosen for test cases. I created simple array to store 16 bytes of key (total of 128 bits of key) and then created round keys using those bytes accordingly.
3. **Add round key:** Before starting 10 rounds, we have to add key to the initial data. So before the “for loop” for 10 rounds, we add (take XOR) key with initial data.
4. Now we come inside the loop, where we will perform byte substitution, row shifting, column mixing and adding round key in this order 10 times. Each step is explained below:
    a. **Byte substitution**: For byte substitution, what we need to do it to read an element from data, separate its first 4 bits and last 4 bits, and replace the original data with data in s-Box in such a way that first 4 bits corresponds to row number of s-Box and last 4 bits corresponds to column number, and then replace the original data with the data in s-Box. I have created a function called “substitution”, which when called by sending a single byte and an index (which shows from which box we want the data to be replaced, either s-Box, or E or L tables which we will use in column mixing), will return the substituted data.

    b. **Row Shifting**: Row shifting is done in such a fashion that first row remains unchanged, in second row each element is given 1 byte circular shift to left, in third row, 2 byte circular shift to left and in fourth row, 3 byte circular shift to left.

    c. **Column Mixing**: In column mixing, the data is multiplied with a constant matrix in Galois Field. The constant matrix is:

    {2,3,1,1}
    {1,2,3,1}
    {1,1,2,3}
    {3,1,1,2}

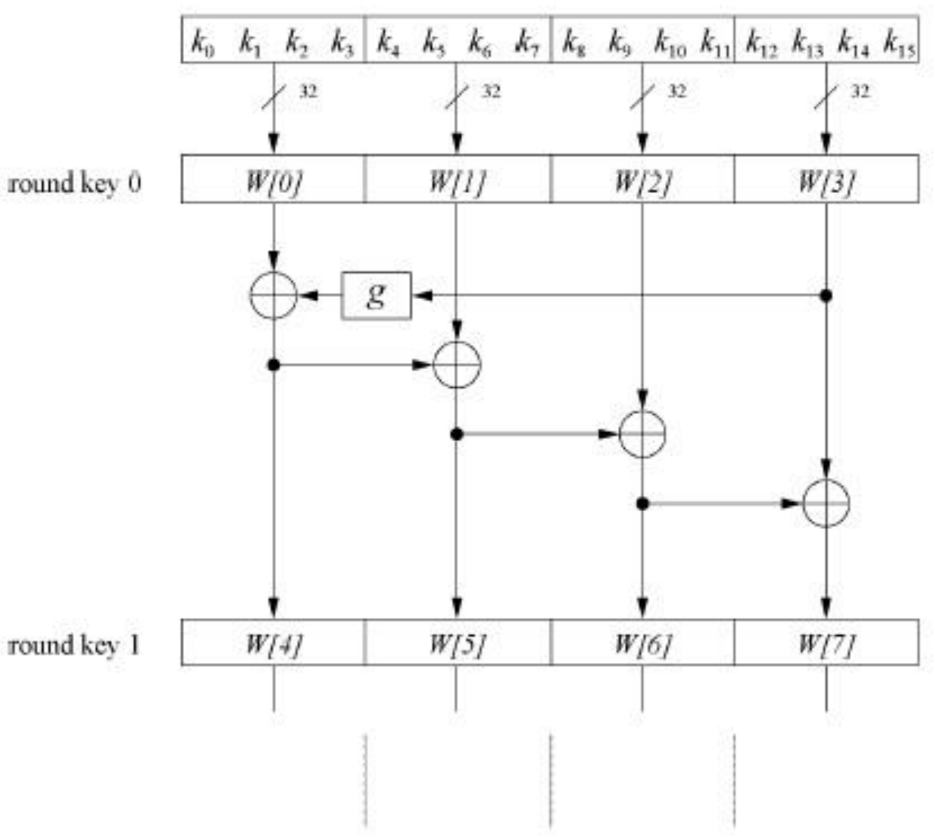
    Multiplication can be done quite easily with the use of the two tables in (HEX), mentioned in the program as "E_Table" and "L_Table". Multiplication is explained here with an example:

    Consider the first column of constant matrix as [2 , 3 , 1 , 1] and then consider a sample input as [0xD4, 0xBF, 0x5D, 0x30]. Keep in mind that the multiplication is done using the tables and the addition is simple bit-wise XOR.

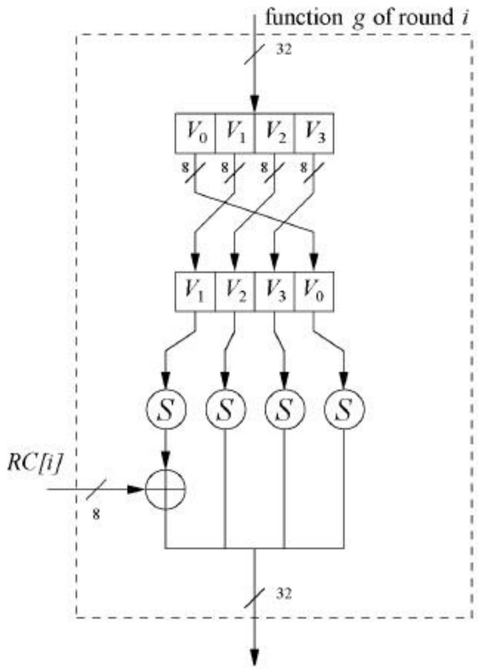
```
[2  3  1  1] x    [D4]
                  [BF]
                  [5D]
                  [30]

Output 0          =    (D4  *   02)  XOR    (BF  *   03)  XOR (5D*1) XOR (30*1)
                  = E (L (D4) + L(02)) XOR E(L(BF) + L(03)) XOR  5D   XOR  30
                  = E (41 + 19)    XOR     E(9D + 01)       XOR  5D          XOR  30
                  = E (5A)         XOR     E(9E)            XOR  5D          XOR  30
                  = B3             XOR DA          XOR  5D           XOR  30
                  = 04
```

And so on for the rest of outputs. First the value is substituted by L_Table and then we take simple arithmetic sum of the values. Now if this sum goes beyond 0xFF, we simply subtract 0xFF from the result, and then using the remaining value, we substitute the value using E_Table. In the end, all the values are XOR'ed and we get the final multiplied value in Galois Field.

d. **Generating round keys**: Round keys are generated by doing XOR in the following way:

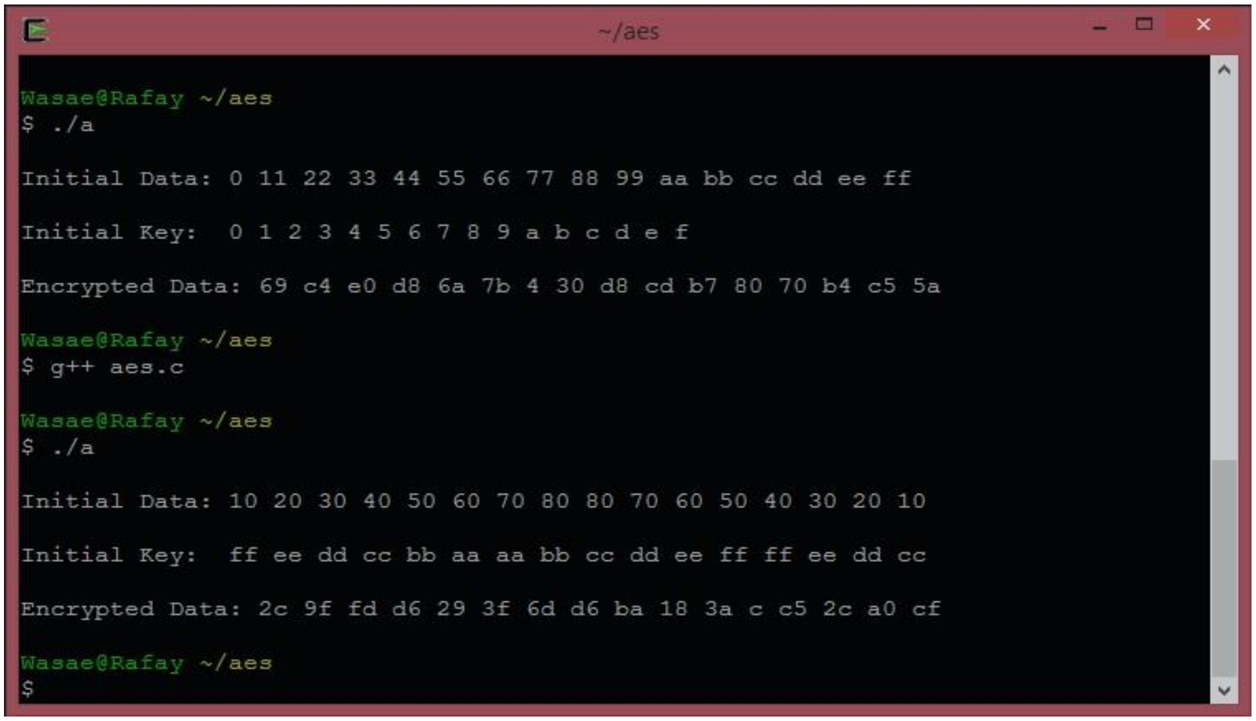

k_0 k_1 k_2 k_3 k_4 k_5 k_6 k_7 k_8 k_9 k_{10} k_{11} k_{12} k_{13} k_{14} k_{15}
32
32
32
32
round key 0
W[0]
W[1]
W[2]
W[3]
g
round key 1
W[4]
W[5]
W[6]
W[7]

So the round key for round 1 will be w[4] – w[7] and in generated from the previous round key. It is generated using simple XOR of each words by the next word, and the function “g” in the small box on top can be explained as:



So the 32 bit word of the key is just swapped in the way it is shown in the figure, and then each byte is substituted from the s-Box and the first byte is XOR'ed with the i^{th} element of RC array ("i" is the number of round and RC is the array of constants which are calculated in Galois Field).

After the program runs for 10 rounds of above 4 functions, we will get the output data as encrypted. Below is a test case for random data taken and random keys used and correspondingly generated encrypted data:



```
Wasae@Rafay ~/aes
$ ./a

Initial Data: 0 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff

Initial Key:  0 1 2 3 4 5 6 7 8 9 a b c d e f

Encrypted Data: 69 c4 e0 d8 6a 7b 4 30 d8 cd b7 80 70 b4 c5 5a

Wasae@Rafay ~/aes
$ g++ aes.c

Wasae@Rafay ~/aes
$ ./a

Initial Data: 10 20 30 40 50 60 70 80 80 70 60 50 40 30 20 10

Initial Key:  ff ee dd cc bb aa aa bb cc dd ee ff ff ee dd cc

Encrypted Data: 2c 9f fd d6 29 3f 6d d6 ba 18 3a c c5 2c a0 cf

Wasae@Rafay ~/aes
$
```

The output can be checked online at http://testprotect.com/appendix/AEScalc