AES ENCRYPTION
FOR 4x4 MATRICES

Emrecan ÜZÜM

First of all, I created two arrays with random generated numbers 0 to 255 for create key and state. Then I took the fourth column of the key array and shifted ones.

```
int keyArr[4][4];
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
        keyArr[i][j] = rand() % 0xFF;
    }
}

int stateArr[4][4];
for (int i = 0; i < 4; i++ ) {
    for (int j = 0; j < 4;j++) {
        stateArr[i][j] = rand() % 0xFF;
    }
}

int RotArr[4];
RotArr[0] = keyArr[3][1];
RotArr[1] = keyArr[3][2];   //rotword
RotArr[2] = keyArr[3][3];
RotArr[3] = keyArr[3][0];
```

After that I put the RotWord in the substitution table and got the output. I also applied XOR operation with 01 as per the rule.

```
RotArr[0] = table[RotArr[0]];
RotArr[1] = table[RotArr[1]];
RotArr[2] = table[RotArr[2]];  //substitute table
RotArr[3] = table[RotArr[3]];



RotArr[0] = RotArr[0] ^ 0x01;
RotArr[1] = RotArr[1] ^ 0x00;  //xor with 1-0-0-0
RotArr[2] = RotArr[2] ^ 0x00;
RotArr[3] = RotArr[3] ^ 0x00;
```

Then i create w4 to w7 with XOR operation between RotWord and W0. With this process, we have obtained w4. We create the w5 and w7 range using the columns we have.

```
    int keyArrTwo[4][4]; //for w4-7
    /*w4*/keyArrTwo[0][0] = RotArr[0]/*rot*/ ^ keyArr[0][0];/*w0*/
    /*w4*/keyArrTwo[0][1] = RotArr[1]/*rot*/ ^ keyArr[0][1];/*w0*/
    /*w4*/keyArrTwo[0][2] = RotArr[2]/*rot*/ ^ keyArr[0][2];/*w0*/
    /*w4*/keyArrTwo[0][3] = RotArr[3]/*rot*/ ^ keyArr[0][3];/*w0*/

    /*w5*/keyArrTwo[1][0] = keyArr[1][0]/*w1*/ ^ keyArrTwo[0][0];/*w4*/
    /*w5*/keyArrTwo[1][1] = keyArr[1][1]/*w1*/ ^ keyArrTwo[0][1];/*w4*/
    /*w5*/keyArrTwo[1][2] = keyArr[1][2]/*w1*/ ^ keyArrTwo[0][2];/*w4*/
    /*w5*/keyArrTwo[1][3] = keyArr[1][3]/*w1*/ ^ keyArrTwo[0][3];/*w4*/

    /*w6*/keyArrTwo[2][0] = keyArr[2][0]/*w2*/ ^ keyArrTwo[1][0];/*w5*/
    /*w6*/keyArrTwo[2][1] = keyArr[2][1]/*w2*/ ^ keyArrTwo[1][1];/*w5*/
    /*w6*/keyArrTwo[2][2] = keyArr[2][2]/*w2*/ ^ keyArrTwo[1][2];/*w5*/
    /*w6*/keyArrTwo[2][3] = keyArr[2][3]/*w2*/ ^ keyArrTwo[1][3];/*w5*/

    /*w7*/keyArrTwo[3][0] = keyArr[3][0]/*w3*/ ^ keyArrTwo[2][0];/*w6*/
    /*w7*/keyArrTwo[3][1] = keyArr[3][1]/*w3*/ ^ keyArrTwo[2][1];/*w6*/
    /*w7*/keyArrTwo[3][2] = keyArr[3][2]/*w3*/ ^ keyArrTwo[2][2];/*w6*/
    /*w7*/keyArrTwo[3][3] = keyArr[3][3]/*w3*/ ^ keyArrTwo[2][3];/*w6*/
```

After this stage, we move on to the second stage and after XOR operation with key and stage, we take the output from the substitution table and create our new array as NewStage

```
    // state ^ key
    int newState[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            newState[i][j] = keyArr[i][j] ^ stateArr[i][j];
        }
    }

    // new state substitute bytes
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            newState[i][j] = table[newState[i][j]];
        }
    }
```

Then we come to the aes shift rows stage and we shifting the rows according to the rule.. And we mixed columns.

```
   //aes shift rows
    int shiftedNewState[4][4];
    shiftedNewState[0][0] = newState[0][0];    //no change in first row
    shiftedNewState[0][1] = newState[0][1];
    shiftedNewState[0][2] = newState[0][2];
    shiftedNewState[0][3] = newState[0][3];

    shiftedNewState[1][0] = newState[1][1];
    shiftedNewState[1][1] = newState[1][2];    //second row => 1 left
    shiftedNewState[1][2] = newState[1][3];
    shiftedNewState[1][3] = newState[1][0];

    shiftedNewState[2][0] = newState[2][2];
    shiftedNewState[2][1] = newState[2][3];    //third row  => 2 left
    shiftedNewState[2][2] = newState[2][0];
    shiftedNewState[2][3] = newState[2][1];

    shiftedNewState[3][0] = newState[2][3];
    shiftedNewState[3][1] = newState[2][0];    //fourth row  => 3 left
    shiftedNewState[3][2] = newState[2][1];
    shiftedNewState[3][3] = newState[2][2];
```

After scrolling, we multiply the array we shifted with our special matrix. The most important point here is that we delete the remaining bits.

```
int tempForXoR[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            tempForXoR[i][j] = shiftedNewState[0][j] * specialMatrix[i][j];
            tempForXoR[i][j] &= 0x00FF; //remove bits after 8 bits
        }
    }
```

After that, we mixing columns.

```
//temp[0][0] ^ temp[0][1] ^ temp[0][2] ^ temp[0][3]
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            shiftedNewState[0][j] = tempForXoR[i][0] ^ tempForXoR[i][1] ^
tempForXoR[i][2] ^ tempForXoR[i][3];
            shiftedNewState[1][j] = tempForXoR[i][0] ^ tempForXoR[i][1] ^
tempForXoR[i][2] ^ tempForXoR[i][3];
            shiftedNewState[2][j] = tempForXoR[i][0] ^ tempForXoR[i][1] ^
tempForXoR[i][2] ^ tempForXoR[i][3];
            shiftedNewState[3][j] = tempForXoR[i][0] ^ tempForXoR[i][1] ^
tempForXoR[i][2] ^ tempForXoR[i][3];
        }
    }

    //we mixed columns. next step is multiplying this state with key w4 to w7.
and we get new state.


    int lastNewState[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            lastNewState[i][j] = shiftedNewState[i][j] ^ keyArrTwo[i][j];
        }
    }
```

The output(encrypted array) of the algorithm is as follows.