Understanding AES Mix-Columns Transformation Calculation

Kit Choy Xintong

University of Wollongong, Year 3 Student

kit_4ever2003@yahoo.com

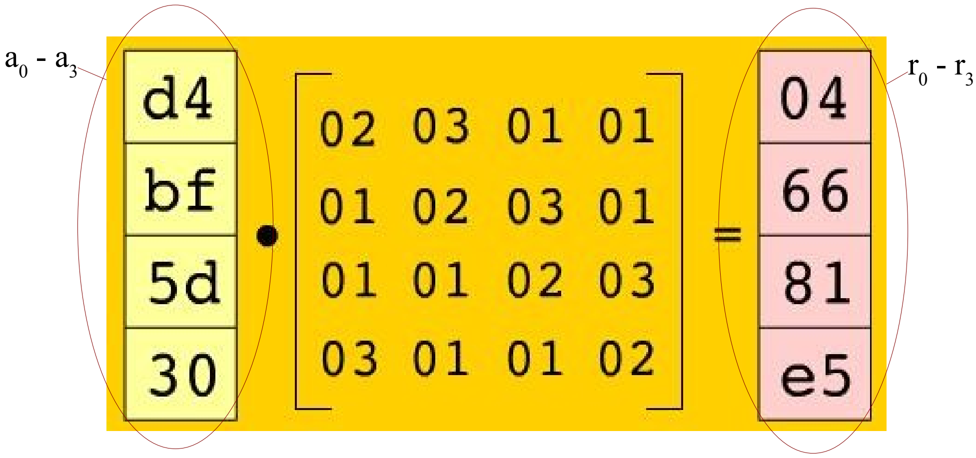
I never really understood the theory behind this when my friend questioned me the other day. Our lecturer actually never really went through it in detail. So we are left to figuring it out ourselves. Just how great is this? Very. Who would have understand them without examples and explained steps? Unfortunately for me, my lecture notes aren't helping me neither is my textbook (Cryptography and Network Security: Principles and Practices by William Stallings). Well, at the least the textbook helps a little more with examples. So here we go:

The mix columns theory is calculated using this formula[1]:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

where r_0, r_1, r_2 and r_3 are the results after the transformation. $a_0 - a_3$ can be obtain from the matrix after the data undergoes substitution process in the S-Boxes. We will now discuss the forward mix column transformation. (I am assuming you know the theory for XOR gates and some other simple theories)

Let's take this example:



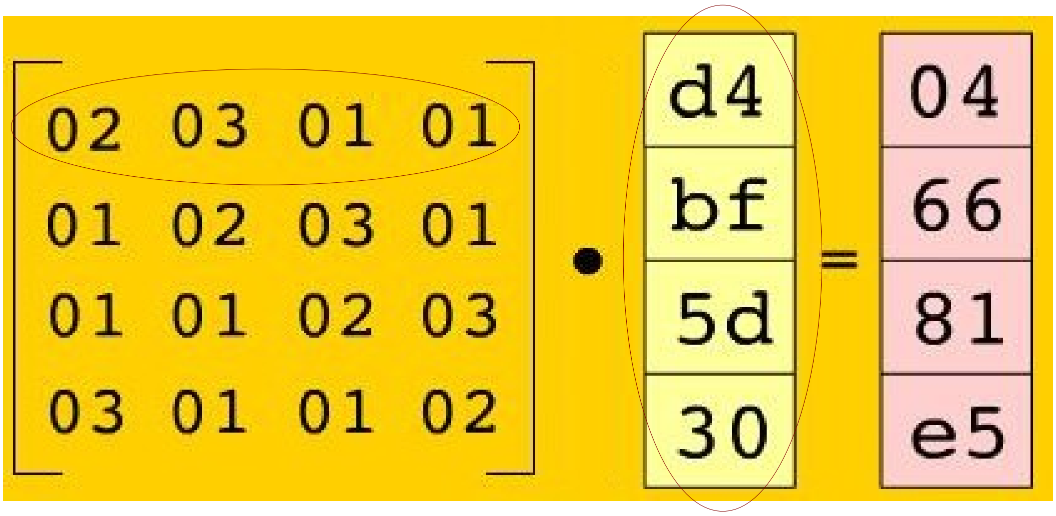
In this example, our $a_0 - a_3$ is equals to d4 – 30 and $r_0 - r_3$ is equals to 04 – e5. One thing to note in this is that it still follows the matrix multiplication rules: row x column. Currently the matrix size looks like this:

$$[4 \times 1] \,.\, [4 \times 4] \neq [4 \times 1]$$

If you would to remember matrix idea of multiplication, to obtain [4 x 1], we need the formula to be

[4 x 4] . [4 x 1] = [4 x 1]

Therefore we need to switch matrices over.



Now we are pretty much ready to calculate the answers. Like I mention early, we will multiply the rows with the column. Let's take the first row of the first matrix and multiply them with our a's values.

To get the r_0 value, the formula goes like this:

r_0 = {02.d4} + {03.bf} + {01.5d} + {01.30}

Wow. Does it not seems easy to obtain the answer? Yes, it LOOKS easy. But when it comes to calculating, apparently it isn't anymore. We will go into the steps one at a time.

1. {02.d4}

We will start with converting d4 to binary. Remember d4 is a byte so when using the Calculator program on the computer, change it to byte under Hex mode. (Qword is usable but I still prefer to change to byte just in case)

d4 = 1101 0100

Now d4 is exactly 8 bits which is good. In the case where you never get a 8 bits long characters such as 25 in Hex (converted: 100101), pad on with 0 in the front of the result until you get 8 characters of 1's and 0's. (25 ends up with 0010 0101)

Now another thing to remember, there is a rule established in the multiplication of the values as written in the book, Cryptography and Network Security[2], that multiplication of a value by x (ie. by 02) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (before the shift) is 1. We can implement this rule in our calculation.

{d4}.{02} = 1101 0100 << 1 (<< is left shift, 1 is the number of shift done, pad on with 0's)
= 1010 1000 XOR 0001 1011 (because the leftmost is a 1 before shift)
= 1011 0011 (ans)

Calculation:

1010 1000
0001 1011 (XOR)
1011 0011

Now we do the same for our next set of values, {03.bf}

2. {03.bf}

Similarly, we convert bf into binary:

bf = 1011 1111

In this case, we are multiplying 03 to bf. Maybe you are starting to wonder how we are going to multiply them, or some might just multiply them directly. For my case, I followed what was suggested in the book[2], we split 03 up in its binary form.

03 = 11
= 10 XOR 01

We are now able to calculate our result.

{03} . {bf} = {10 XOR 01} . {1011 1111}
= {1011 1111 . 10} XOR {1011 1111 . 01}
= {1011 1111 . 10} XOR {1011 1111}
(Because {1011 1111} x 1[in decimal] = 1011 1111)
= 0111 1110 XOR 0001 1011 XOR 1011 1111
= 1101 1010 (ans)

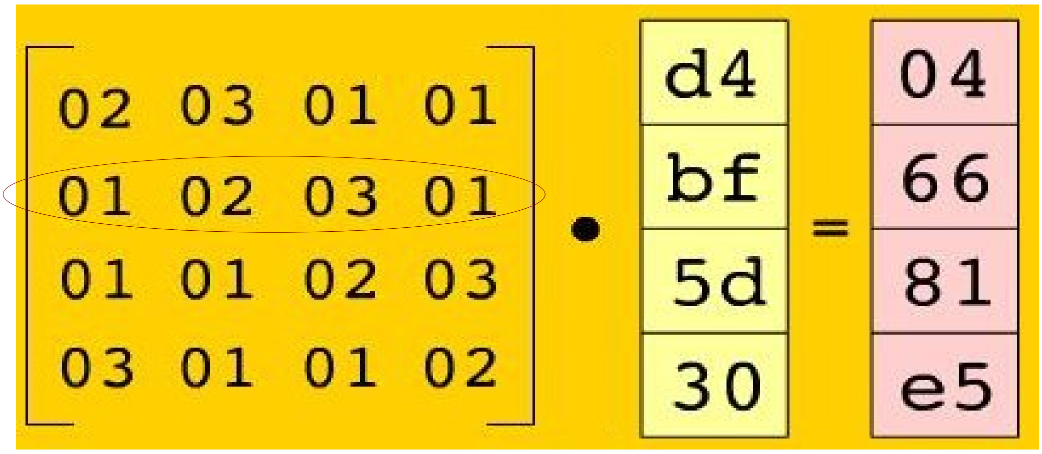
{01.5d} and {01.30} is basically multiplying 5d and 30 with 1(in decimal) which we end up with the original values. There isn't a need to calculate them using the above method. But we do need to convert them to binary form.

5d = 0101 1101
30 = 0011 0000

Now, we can add them together. As they are in binary form, addition will be using XOR.

r_0 = {02.d4} + {03.bf} + {01.5d} + {01.30}
= 1011 0011 XOR 1101 1010 XOR 0101 1101 XOR 0011 0000
= 0000 0100
= 04 (in Hex)

Let's try the next row.



r_1 = {01.d4} + {02.bf} + {03.5d} + {01.30}

1. {02.bf}

{bf} . {02} = 1011 1111 << 1
= 0111 1110 XOR 0001 1011
= 0110 0101

2. {03.5d}

{5d} . {03} = {0101 1101 . 02} XOR { 0101 1101}
= 1011 1010 XOR 0101 1101
= 1110 0111

Therefore,

r_1 = {01.d4} + {02.bf} + {03.5d} + {01.30}
= 1101 0100 XOR 0110 0101 XOR 1110 0111 XOR 0011 0000
= 0110 0110
= 66 (in Hex)

We got our second values, 66. Do the same for the rest and you will get all the results. We now know how to calculate the mix columns. :) Happy calculating~!

References:

[1] Wikipedia – Rijndael mix columns, [Online]
Available: http://en.wikipedia.org/wiki/Rijndael_mix_columns

[2] William Stalling (2006), Chapter 4.6 Finite Fields of the Form $GF(2^n)$ – Multiplication, in Cryptography and Network Security: Principles and Practices, Page 125 – 126.

PS. There is something call the Inverse Mix Column Transformation. But I don't think I will be touching it right now.