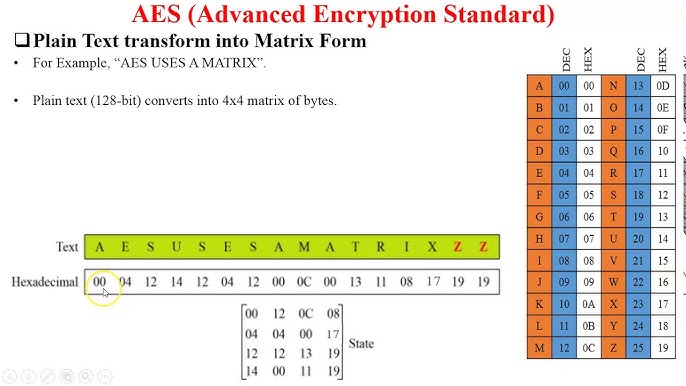
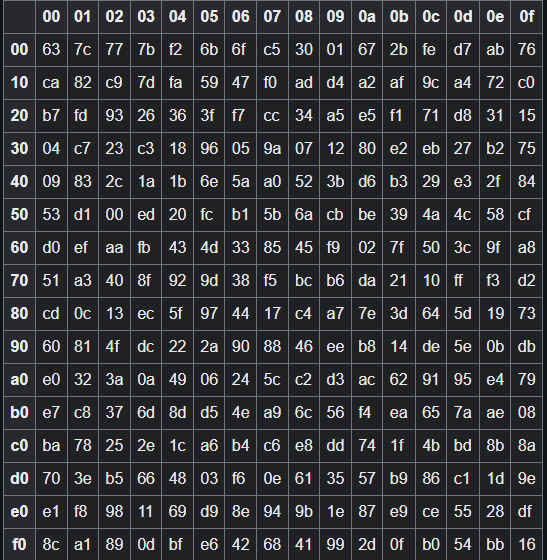
Encryption & Decryption

**Overview:**

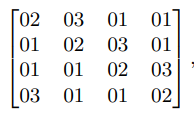
* The project follows the principle of AES (advanced encryption system). This system turns a data of 16 bytes (in this case a string of 16 characters) into a 4x4 matrix.



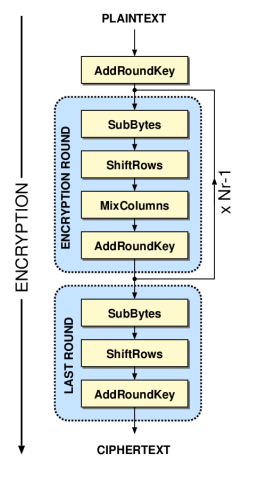
* Like all encryption system keys this also utilizes round keys; a predefined key gets edited/changed with each round of the encryption.
* An encryption can be 128, 192 and 256 bits. 128 bits takes 10 rounds, 192 takes 12 and 256 takes 14.
* This matrix will have its value substituted by a predefined sub box



* The rows are of the matrix (now will be referred to as “state”) will shift according to the index of the row i.e. if index of the row is 0 then it will shift zero times to the left; however, if the index is 1 then the values will shift by one to the left and the value that will move out of the matrix will return to the last place of the row.
* The column will then be mixed by multiplying with a predefined matrix, whose inverse does exist



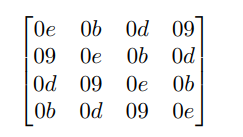
* The column mixing is done by using Galois field, the basic concept of the field is in this case is to not leave the range of 1 byte (0-255) and all operations like addition are done with XOR. To oversimplify it, it works like analogue clocks, when the hand reaches 12 you then go to 1 or if the hour hand is at 11 and you have to add 2 hours you would reach 1.

[](#_top)

Figure

<- This diagram encapsulates the operations of the code.

* The decryption is a mirror version of the above-mentioned process.
* First the round keys will add.
* The values for the reverse column mixing are the following:



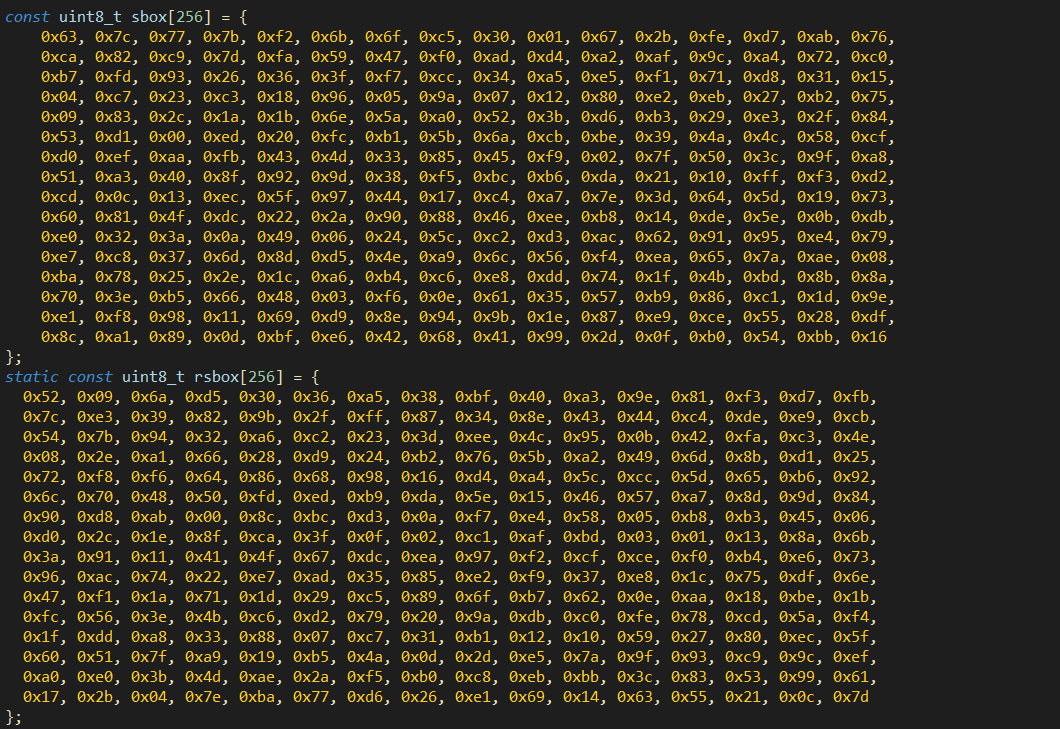
* Rows will be shifted exactly as mentioned before.

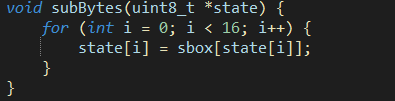


* This is the table that represents the reverse of the sub box.
* This process will be repeated as many times as encryption.

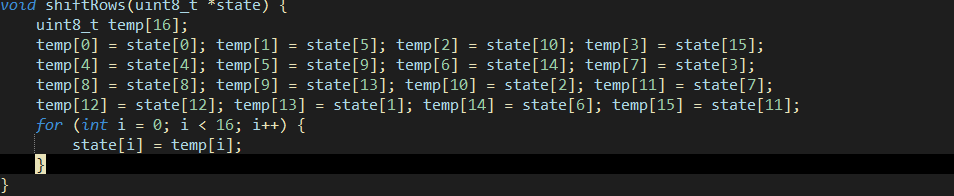
**The Code:**

For this code we will be using uint8\_t from stdint library, to ensure that our data will be dealt in 8 bits.

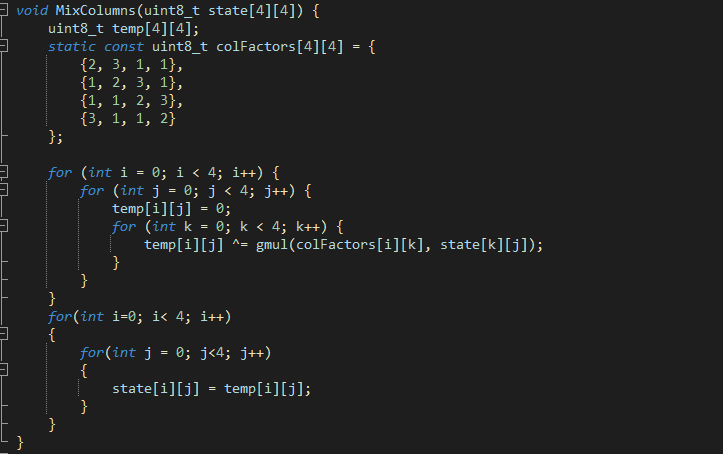
****

* These arrays represent the contents of the substitution boxes
* This function will take the current state and of the data and substitute it appropriately.

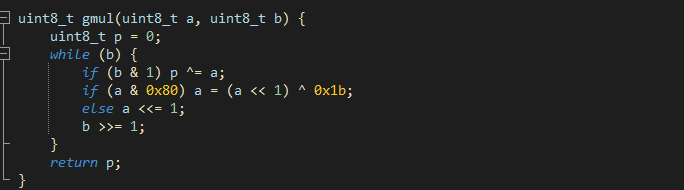
i.e. if the “A” must be substituted it will be substitute by sbox[41] since “A” is 0x41 in hexa



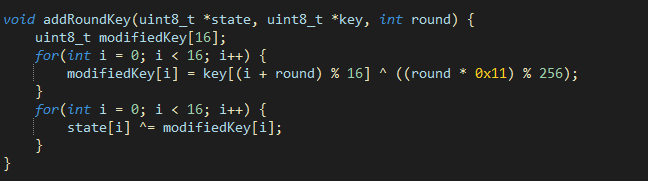
* This is a simple code to shift rows appropriately.



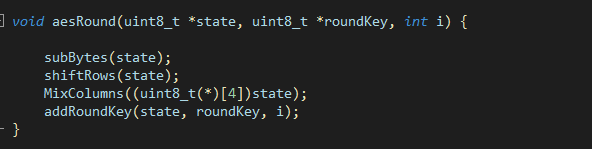
* This function handles the column mixing by incorporating the gmul function which utilizes the above-mentioned Galois Field.
* Here is the gmul function



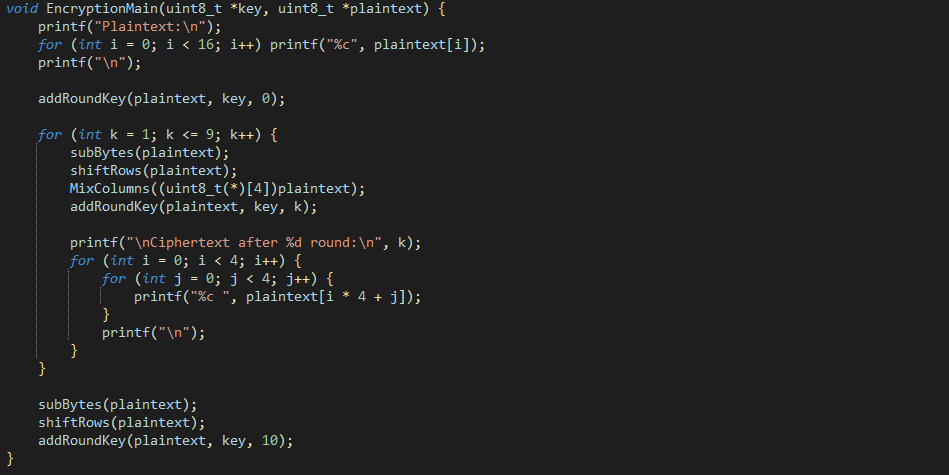
The formula is very simple. A local variable is first initialized by zero, and a loop will iterate till the second number of the multiplication is a non-zero. In the loop, if the least significant bit is 1 then a simple XOR operation will be commenced on the variable with the first value. The second if statement deals with the overflow of the most significant bit, if it occurs then the bits will be shifted to the left and a XOR operation will be done with 0x1b. We chose 0x1b because it is an irreducible number of the Galois Field. If there is no overflow then a simple bit shifting is done. Throughout the loop, one thing remains constant: shifting b towards the right side. This whole operation outputs the result of p.



* This function handles the addition of the round keys. A simple formula was designed to handle this function and the XOR operation done in the first operation is there to check for any overflow values.

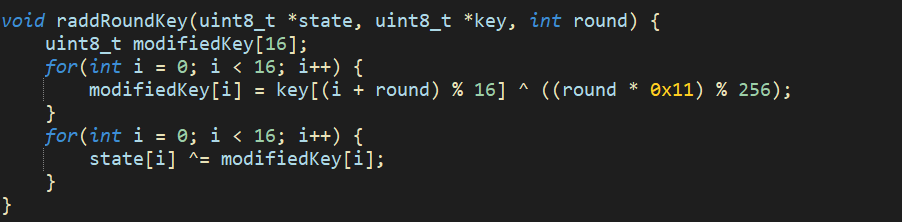


* All of the functions above are concised in this one function to be called in a loop. But this was later scrapped because initially only the round key is first added so every function has to be called independently and hence removed from the main functions.

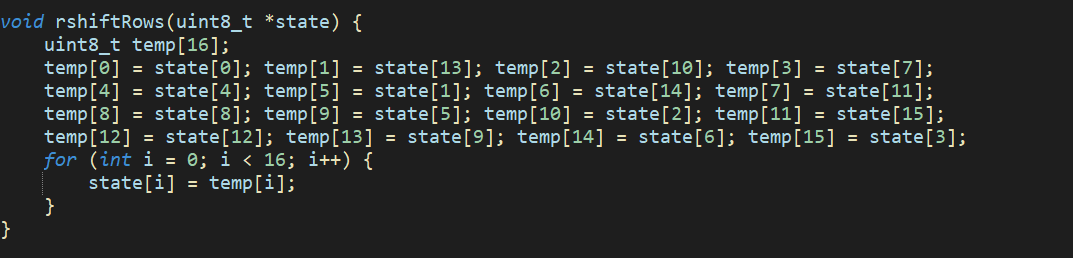


* Encryption main function follows Figure 1.

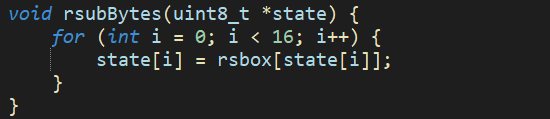
The decryption process is no different but rather mirrored version of it. Understanding encryption by itself was enough to grasp the concept of decryption and it’s approach



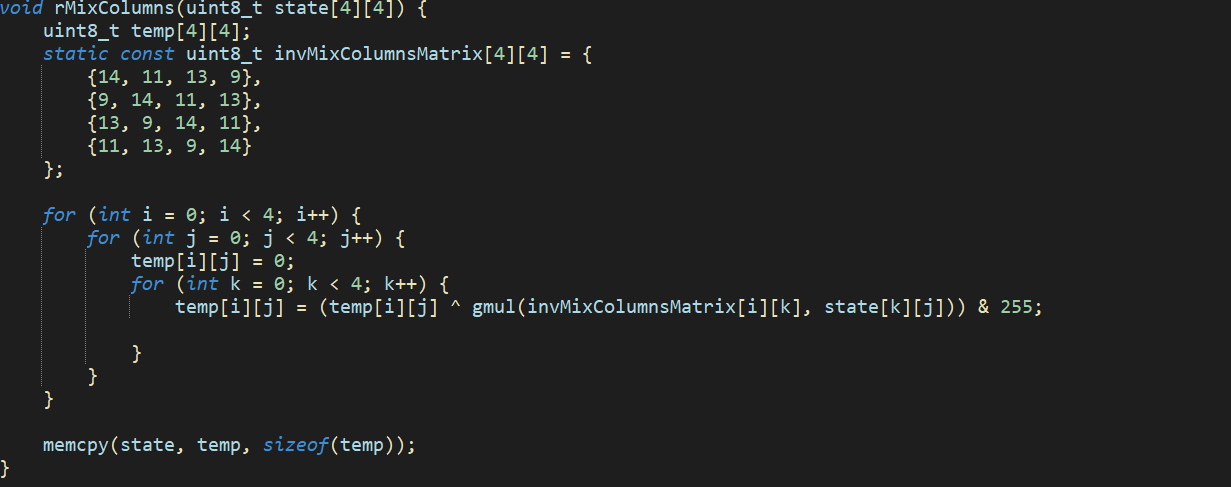
* Reverse of the addition of the key.



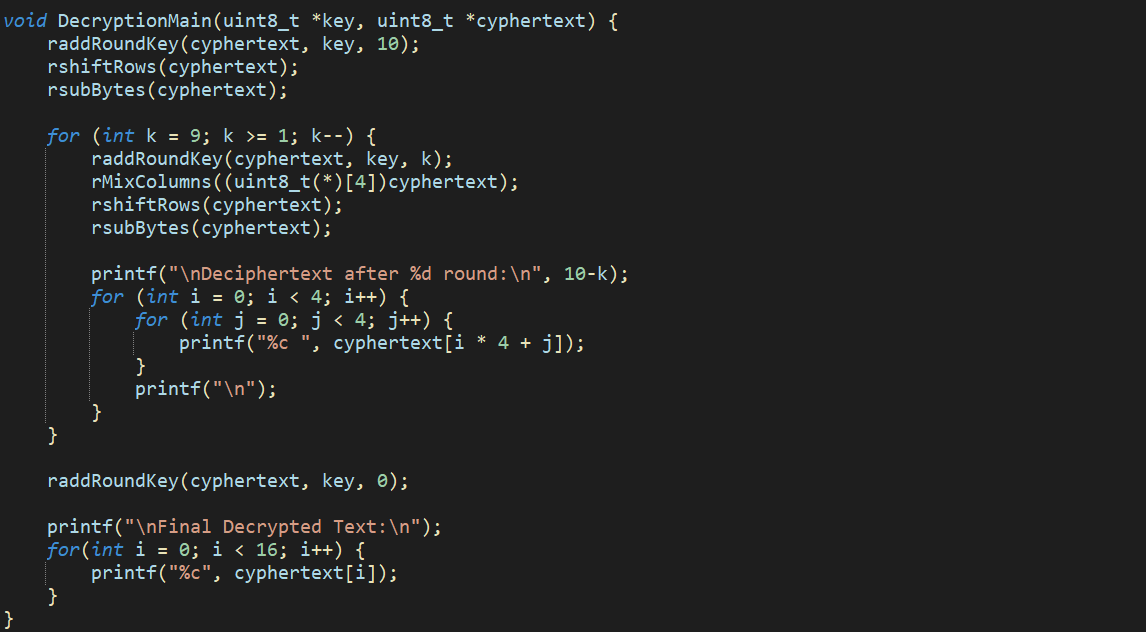
* Then comes the reversal of the row shifting



* These values are then substituted back with the help of the reverse sub box

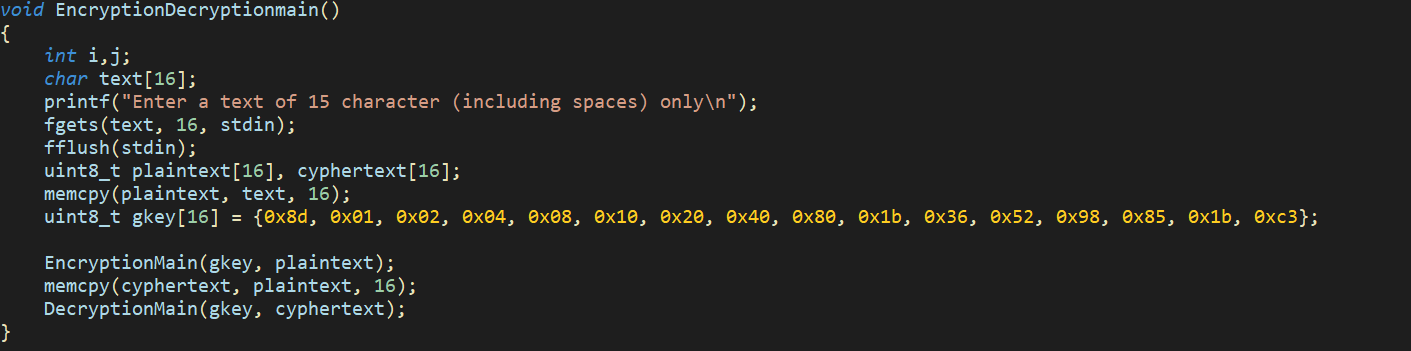


* After that comes reverse mixing of the column matrix



* Finally all of it is then called in the Decryption main function

The following function is called in the main, this handles everything up until now. It takes input from the user and copies the contents of the memory in a variable which is then passed through the functions.



Issues Faced:

* At first a function was used to generate random keys, but due to time constraints and the complexity of the project it was then removed.
* Another function was first made to make a column shuffling generator but that was also scraped since not all values had inverses.
* Initially, before the creation of the galois field, the module was not correctly decrypting the code.

**Strengths:**

* If the key is safe and the round key aswell, it is very difficult to crack.
* The galois field is a powerful tool, since it is efficient and provides an acceptable encryption.
* Since everything is in a sub box (also known as a lookup table), everything works smoothly and efficiently.
* Since the key changes each iteration it is almost impossible to get to the original key without the algorithm to add round keys.

**Limitation:**

* A text must be up to 15 characters (1 character for null character).
* The algorithm is very basic since this is a demonstration, the key is much shorter and only does 10 rounds.

Resources Used:  
<https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

<https://github.com/kokke/tiny-AES-c/tree/master>

<https://en.wikipedia.org/wiki/Rijndael_S-box>

<https://crypto.stackexchange.com/questions/2418/how-to-use-rcon-in-key-expansion-of-128-bit-advanced-encryption-standard>

<https://www.youtube.com/watch?v=O4xNJsjtN6E&ab_channel=Computerphile>