Benjamin Kwon

Advanced Encryption Standard

In this code, input must be manually added into the corresponding *unsigned char* **Plaintext** and **Key** *array*. There are currently two test codes that are in the C++ file with one segment of the test code commented out. These variables and test codes reside in the main which calls the *KeyExpansion* first and *AES* methods second.

KeyExpansion(unsigned char key[16], unsigned char Expandedkey[176]){...};

The code starts off adding the four words or 16 bytes of the user inputted *key* into the first 16 bytes of the *ExpandedKey*. After doing so, there is a for loop that will fill up the remaining indexes of the *ExpandedKey*. There are four *unsigned char temp* variables that will hold the previous four bytes of the *ExpandedKey* after the definition of those four bytes. Usually in this method, the indexes are filled by XOR from previous indexes but the only occasion when it relied on modified bits from previous indexes was every four words, thus there is an if statement with many OR statements. In this if-statement, there is a total of three methods that are later diffused together: *RotWord*, *Subword*, and *Rcon*. *Rotword* simply shifts the word or 4 separate bytes of unsigned char, in this case, once to the left. Subword takes the resultant of *Rotword* and substitutes the bytes in the *temp* from the Rjindael S-box. This is implemented by getting the numerical or decimal number of the hex that resides in the *unsigned char temp* and using that to find the index array of the Sbox, rather than having a row-column array. The last method *Rcon* simply takes the first byte from the word and XOR the byte and from an round constant array that consists of 11 different types of hexadecimal.

AES(unsigned charPT[16], unsigned char EK[176])//PT = PlainText && EK = ExpandedKey

In this method, the 10 rounds of an 128 bit input plaintext and key is invoked with some minor changes, according to the Rjindael's AES. The code, at first, calls the first *AddRoundKey*, and then loops 9 times of *SubBytes, ShiftRows, Mixcolumns, AddRoundKey* in this order and ends the method with *SubBytes, ShiftRows, AddRoundKey*. Because the ExpandedKey is actually the entire 176 bytes instead of 44 words, it is manually incremented to represent 4 words.

AddRoundKey(unsigned char PT[16], unsigned char EK[176], int Position)

This method is merely just a for loop that does a logical operation Exclusive-or onto each of the PlainText byte and the corresponding ExpandedKey byte that is incremented through Position.

SubBytes(unsigned char PT[16])

This method is simple because it is a for loop that checks the decimal value of the value in the PlainText byte. This decimal value serves as the index of the Rjindael S-box and the value inside of this index is the hexadecimal value that the PlainText byte will keep.

ShiftRows(unsigned char PT[16])

ShiftRows is self-explanatory method that requires shifting each row to a specific amount except the first row[0-3] bytes. The second row[4-7] is a circular shift to the left once. The third row[8-11] is a circular shift to the left twice. And finally, the fourth row[12-15] is a circular shift to the right once.

MixColumns(unsigned char PT[16])

This method involves understanding the Rjindael's Galois field. Although we do multiply and add it like any other matrix, the multiplication and addition have different set of mathematical rules than the common ones. These different mathematical operations resides in Rjindael's Galois field or **GF**(2^8). To summarize this method, it manipulates bits resulted from the multiplication between hexadecimal values and values from the **GF** and adds all of the resultants into one hexadecimal value which becomes the PlainText byte.