CSE 178 - Network Security

AES Encryption Implementation

Jonathan Chancey

Introduction

AES, also known as Rijndael (pronounced RAIN-dahl), is a block cipher that encrypts 16 bytes of data at a time. If the message is not 16 bytes in length, it will be padded with zeros. Although the original algorithm supports three different key lengths—128, 192, and 256 bits—my implementation uses the most common key size, 128 bits, and thus has 10 rounds of encryption. AES is made up of rounds. Round keys are keys made using the cipher key after it undergoes the key expansion routine. S-Box is a non-linear substitution table that's used byte substitution and key expansion in order to achieve parity in byte-value substitution.

Code Explanation

There are three main functions in the AES algorithm. SubBytes, ShiftRows, MixColumns, and AddRoundKey. These will be explained first due to their importance.

SubBytes is a relatively simple function. It simply populates the state array with information from the Rijndael S-Box lookup table.

```
void SubBytes(unsigned char* state){
    for(int i = 0; i < 16; i++){
        state[i] = s_box[state[i]];
    }
}
```

ShiftRows cyclically shifts the last three rows of the state. This is a concept best illustrated. Say it was given an input of 16 numbers numbered 1-16 for simplicity. Arranged in a 4 by 4 matrix the first row is shifted left by 0, the second by 1, the third by 2, and the fourth by 3.

```
void ShiftRows(unsigned char* state){
    unsigned char tmp[16];
    tmp[0] = state[0];
    tmp[1] = state[5];
    tmp[2] = state[10];
    tmp[3] = state[15];

    tmp[4] = state[4];
    tmp[5] = state[9];
    tmp[6] = state[14];
    tmp[7] = state[3];
    tmp[8] = state[8];
    tmp[9] = state[13];
    tmp[10] = state[2];
    tmp[11] = state[7];

    tmp[12] = state[12];
    tmp[13] = state[1];
    tmp[14] = state[6];
    tmp[15] = state[11];

    for(int i = 1; i < 16; i++){
        state[i] = tmp[i];
    }
}
```

Before					After			
1	2	3	4	shifts 0	1	2	3	4
5	6	7	8	shifts 1	8	5	6	7
9	10	11	12	shifts 2	11	12	9	10
13	14	15	16	shifts 3	14	15	16	13

MixColumns uses a Galois multiplication field, dot products, and the XOR operator to mix the data of all the columns in the state. This mathematical technique is used every input so that every input creates a unique output that is the size of or smaller than 1 byte.

```
void MixColumns(unsigned char* state){
    unsigned char tmp[16];

    tmp[0] = (unsigned char)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);
    tmp[1] = (unsigned char)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);
    tmp[2] = (unsigned char)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);
    tmp[3] = (unsigned char)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

    tmp[4] = (unsigned char)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);
    tmp[5] = (unsigned char)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);
    tmp[6] = (unsigned char)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);
    tmp[7] = (unsigned char)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

    tmp[8] = (unsigned char)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);
    tmp[9] = (unsigned char)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);
    tmp[10] = (unsigned char)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);
    tmp[11] = (unsigned char)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

    tmp[12] = (unsigned char)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);
    tmp[13] = (unsigned char)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);
    tmp[14] = (unsigned char)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);
    tmp[15] = (unsigned char)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

    for(int i = 0; i < 16; i++){
        state[i] = tmp[i];
    }
}
void AddRoundKey(unsigned char* state, unsigned char* roundKey){
    for(int i = 0; i < 16; i++){
        state[i] ^= roundKey[i];
    }
}
```

AddRoundKey is called after the key is expanded. This is the most basic function of the entire file. It simply adds the state equal to the bitwise XOR of the roundKey array.

```
void AddRoundKey(unsigned char* state, unsigned char* roundKey){
    for(int i = 0; i < 16; i++){
        state[i] ^= roundKey[i];
    }
}
```

Walkthrough

The program starts by declaring the message and the key. If the message is not divisible by 16, it is padded with zeros. AES_Encrypt is then used to encrypt blocks of 16 bytes. AES_Encrypt is the main encryption function that takes the padded message and the key as parameters and then calls the functions of the AES algorithm. The message is copied into the state. Next, KeyExpansion generates a series of round keys and the Initial Round begins with AddRoundKey being called. Now that the initial functions have been called, the loop is started. SubBytes, ShiftRows, Mix Columns, and AddRoundKey are now called equal to the numberOfRounds. Ten rounds of encryption are used as this is an implementation of AES-128. Due to the final round being run outside of the loop, numberOfRounds is set to 9. After the loop ends, the final round begins which calls SubBytes, ShiftRows, and AddRoundKey, but does not call MixColumns. The state is then copied to message signifying the end of the encryption process. With the help of PrintHex—a simple function that takes an unsigned character and prints its hex value—the original message and encrypted message are printed to the console and the program ends.

This AES implementation is written in C and can encrypt messages of various lengths with a specified key and output its corresponding ciphertext.

Citations:

https://github.com/bighil/aeslua/tree/master/src/aeslua
http://www.goliathdesigns.com/2009/12/aes/
https://www.youtube.com/watch?v=K2Xfm0-owS4
http://www.goliathdesigns.com/wp-content/uploads/files/aes/fips-197.pdf

http://wiki.c2.com/?DesignBeforeCoding
http://wiki.c2.com/?SmallIsBeautiful

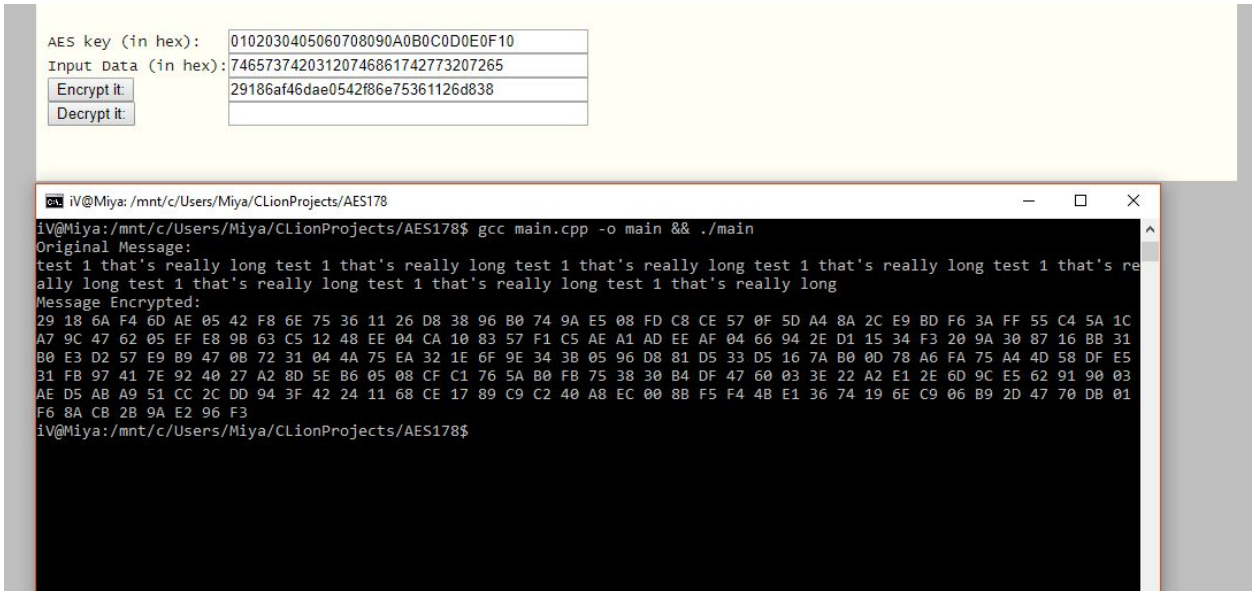
http://testprotect.com/appendix/AEScalc
http://www.unit-conversion.info/texttools/hexadecimal/

Appendix:

Test case 1

Input: “test 1 that’s really long”

Output: ”29 18 6A F4 6D AE 05 42 F8 6E 75 36 11 26 D8 38 96 B0 74 9A E5 08 FD C8 CE 57 0F 5D A4 8A 2C E9 BD F6 3A FF 55 C4 5A 1C A7 9C 47 62 05 EF E8 9B 63 C5 12 48 EE 04 CA 10 83 57 F1 C5 AE A1 AD EE AF 04 66 94 2E D1 15 34 F3 20 9A 30 87 16 BB 31 B0 E3 D2 57 E9 B9 47 0B 72 31 04 4A 75 EA 32 1E 6F 9E 34 3B 05 96 D8 81 D5 33 D5 16 7A B0 0D 78 A6 FA 75 A4 4D 58 DF E5 31 FB 97 41 7E 92 40 27 A2 8D 5E B6 05 08 CF C1 76 5A B0 FB 75 38 30 B4 DF 47 60 03 3E 22 A2 E1 2E 6D 9C E5 62 91 90 03 AE D5 AB A9 51 CC 2C DD 94 3F 42 24 11 68 CE 17 89 C9 C2 40 A8 EC 00 8B F5 F4 4B E1 36 74 19 6E 85 EA C3 00 22 36 AF E9 EE 8A 0D A8 01 A7 77 4B”



Test case 2

Input: “This is an AES block cipher test”

Output: “7D 00 91 53 13 2E 20 C5 08 1C 77 70 BA 39 95 7F 6E CF D6 49 FC 54 58 F3 07 A0 E7 4F EF B1 CC B1”