Advanced Encryption Standard Implementation

For 128, 192 and 256 bits

Course ID: MSCS 630
Course Name: Security Algorithms and Protocols
Professor: Pablo Rivas-Perea
Author: Sandeep Reddy Salla, Himaja Kethiri
Submitted Date: May 2, 2016

Contents

1. Introduction

Advanced Encryption Standard is a symmetric block cipher that can process the 128bit data using key length of 128,192 and 256 bits. This document describes the encryption and decryption process of AES. In encryption process 128-bit block cipher is encrypted using key length of 128bits.In the standard, the encryption algorithm is referred to as the cipher and the decryption algorithm as the inverse cipher.

2. Program Interface

The main aim of this program is to provide an encryption and decryption operation. This program accepts input from the console or from the input file.

Accept Input from Console

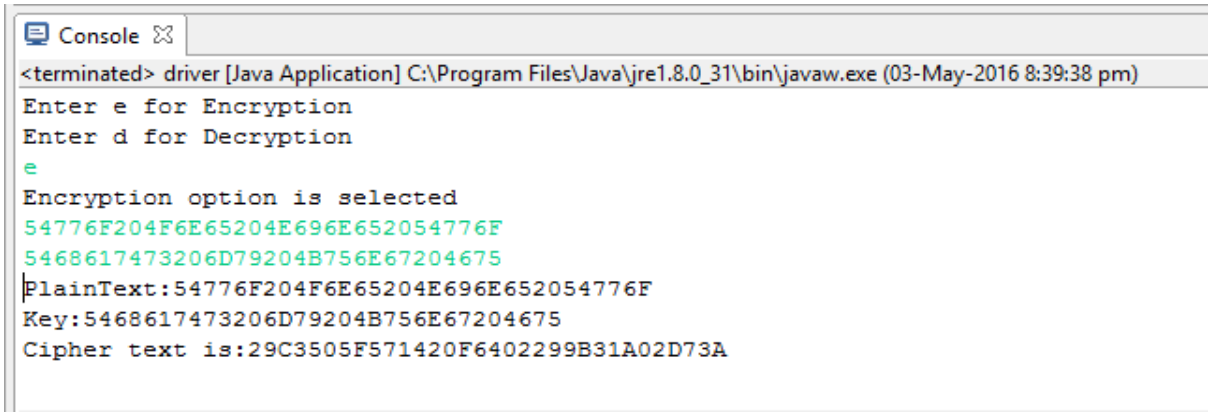
- Run the program.
- Select the option from the console either to select encrypt or decrypt.
- If we select encryption, enter "e" and for decryption enter "d".
- Based on the selection provide valid input like plaintext and key for encryption.
- Cipher text result will be displayed.

Running the program from the console

- Compile the java program from the command prompt.
- Once the compilation is successful, give an input file consisting of option selected either "e" or "d" along with the plaintext/ciphertext and the key.
- The output is displayed on the command prompt.

3. Screenshots

```
D:\JavaWorkSpace\H-AES\src>java Driver < test.2.txt
Enter Encrypt for e
Enter Decrypt for d
Decryption option is selected
CipherText:A52D12429C45282D4DF3682363713FEB
Key:5F72641557F5BC92F7BE3B291DB9F91A
Plain text is:63CAB7040953D051CD60E0E7BA70E18C
D:\JavaWorkSpace\H-AES\src>java Driver < test.1.txt
Enter Encrypt for e
Enter Decrypt for d
Encryption option is selected
PlainText:63CAB7040953D051CD60E0E7BA70E18C
Key:5F72641557F5BC92F7BE3B291DB9F91A
Cipher text is:A52D12429C45282D4DF3682363713FEB
D:\JavaWorkSpace\H-AES\src>
```



4. Input and Output

Format of Input and Output

- For 128-bit AES encryption and decryption, the length of the hexadecimal plaintext, ciphertext and the length of the key must be equal to 32bits.
- For 192-bit AES encryption and decryption, the length of the hexadecimal plaintext, ciphertext must be equal to 32 bits and the length of the key must be equal to 48 bits.
- For 256-bit AES encryption and decryption, the length of the hexadecimal plaintext, ciphertext must be equal to 32 bits and the length of the key must be equal to 60 bits.

We can also read input from input files and can be redirected to the output file. For performing Encryption operation, provide the inputs like selecting an option, plaintext and the key.

Example

In the example a, we are performing Encryption Operation by providing plaintext and key to get the ciphertext as output.

a. Encryption operation

Input: 63CAB7040953D051CD60E0E7BA70E18C (Plain Text)
5F72641557F5BC92F7BE3B291DB9F91A (Key)

Output: A52D12429C45282D4DF3682363713FEB (Cipher Text)

In the example b, we are performing Decryption Operation by providing ciphertext and key to get the original plain text as output.

b. Decryption Operation

Input: A52D12429C45282D4DF3682363713FEB (Cipher Text)
5F72641557F5BC92F7BE3B291DB9F91A (Key)

Output: 63CAB7040953D051CD60E0E7BA70E18C (Plain Text)

5. Program Structure

Encryption

AESEncrypt.java

In the encryption process different methods are used to encrypt the data.

1. aesRoundKeys()
2. aesNibbleSub()

3. aesShiftrows()
4. aesRcon()
5. aesSbox()
6. computeXOR()
7. getWmatrix()
8. aesMixColumns()
9. aes()

aesRoundKeys() will take input as the plaintext which is the 4*4 matrix and it will split the string for every 2 bytes and stores in the matrix format.

aesNibbleSub() will take input from the aesRoundKeys() and it will substitute the values with the corresponding values from the S-BOX lookup table.

aesShiftrows() will take input from the aesRoundKeys() and performs the left shift operation .

aesRcon() calculates the round value and based the round value, the value corresponding to the round will fetched from the R-CON lookup table. It will be used to generate 4*44 matrix.

aesSbox()-It will take input as the hexadecimal and converts the values into integer and send as the input to the S_BOX to fetch the values.

getWmatrix() will be used to generate the 4*44 matrix. In this 4*44 matrix will be generated based on some conditions. WMatrix [][] is the 4*44 matrix that can be generated using previous column.

I. If the column (j) value in WMatrix [j] is not multiple of 4. Then we will XOR the values based on below equation.
   W[j] =W [j-4] ^ W [j-1]
II. If the value of the column(j) is multiple of 4.Then
   A. Perform the transpose of the column matrix and store it in temporary column matrix.
   B. Then perform the left shift operation on newly generated matrix
   C. Transform the each byte of the matrix using S-BOX
   D. Perform the XOR operation using corresponding round constant(Rcon(i)) that calculated using aesRcon() method
   E. Finally calculate the W[j] using,

$$W[j] = W[j-4] \wedge W_{NEW}$$

Similar logic is used for WMatrix for 192 and 256-bits but with the change of column values.

computeXOR() it will XOR the bytes . It can be used in getWmatrix () method to compute the XOR of the new matrix with rcon (value fetched from the lookup table) value.

aesMixColumns() In this method we are using Mul2, Mul3 look-up table that are introduced by Rijndael. It also makes use of Hash maps to map the values to the lookup tables. These lookup tables are having constant values that can be used to XOR the values.

Aes() - In this method we are performing the stateXOR(), NibbleSub(), ShiftRows() and MixColumns() based on the round. For the last round MixColumns () will not be used. In this method cipher text will be produced.

Decryption

AESDecrypt.java

In Decryption process we are providing the input as cipher text that produced in encryption. The key is used to decrypt it. The main goal of decryption is to reproduce the plain text using both the cipher text and key.

In this we are using following method.

1. MatrixSelection()
2. aesRoundKeys()

3. InvNibbleSub()
4. InvShiftrows()
5. aesRcon()
6. aesSbox()
7. computeXOR()
8. getWmatrix()
9. InvMixColumns()
10. InvAes()

MatrixSelection()-This method facilitates to select the length of the key and the matrix should be selected based on the value provided in the column Size variable. We are initializing the number of rounds based on the key length 128, 192, and 256-bits.

aesRoundKeys()-This method takes 4*4 key matrix as input and splits it for every 2 sub bytes. In this we are giving input in the form of string matrix.

InvNibbleSub ()-This method takes input from aesRoundKeys and fetches the corresponding values from INV_SBOX and substitutes it.

InvShiftrows() - This method performs Right shift on the key matrix and stores it in to the input matrix.

aesRcon() -This method calculates the round value and based on that round it fetches rcon (round constant) corresponding round value from the R_CON table which will be used in the getWmatrix() to generate the 4*44 matrix.

aesSbox()- This method fetches the corresponding S-Box value from the lookup table. Before that it converts the hexadecimal value in to integer.

computeXOR ()- This method will take as two hexadecimal values and performs XOR operation on those values. If the length of the value is 1 then it appends '0' to it.

getWmatrix()- This method will take the input as 4*4 matrix and generates 4*44 based on the WMatrix[][] conditions that we have used in encryption getWmatrix().In this we every time we are going to decrement the column value. Every column can be calculated by using the previous column.

InvMixcolumn ()- This method takes input from InvShiftRows() and performs the mixcoulmns operations by using mul9,mul11, mul13, mul15 look-up tables. These values can be mapped or get from the lookup tables based on hash maps. The corresponding values get XORed with the values present in the input matrix.

InvAes ()-This method will take input matrix and performs series of operations based on the round. For the first and last round it will perform all operations like (InvNibbleSub (), InvShiftrows, and except InvStateXOR () except InvMixcolumn ().Remaining all rounds makes use of all operations. In this method we are going to produce the plaintext.

Working of Driver file

The driver file accepts input in two ways.

- Using standard input
- Using test cases (text file)

1. We can give the input using standard input that is using command prompt by entering the Either plain or cipher text and key.
2. Using test cases (text file)-In this we are giving the input using text file. The file should contain 3 lines.

In first line we are giving option to select 'e' for encryption and'd' for decryption. Next the plain text (or) cipher text can be given based on the encryption (or) decryption strategy we choose. Lastly

we will give the input key (128, 192 and 256-bits) based on its length. The driver file will take the text file as input and produces the output on the console.

6. Improvements and Extensions

Improvements: Extension of encryption algorithm from 128-bit to 192 and 256-bits
Decryption algorithm for 128, 192, and 256-bits

Padding Strategy: Zero padding can be used to make the plaintext suitable to make it a 128-bit. But, zero padding is not reversible. It's better not to use the zero padding strategy.

A simple padding technique has been used for the text that are not of suitable length.

1. First determine the length of the text (plain text or cipher text) to find the number of bits need to pad.
2. The number of bits (n) required and the length of the text should be multiple of 32 (128-bits).
3. Then the text will be padded n times with n values.

The reason behind choosing this padding strategy is that, it is very simple and easy to implement. We tried this padding for 128-bits. But, we failed to implement it perfectly as we were new to this technology.

Authentication Strategy:

- SHA-3 will be the best authentication strategy to be used to avoid the unauthorized modifications to the messages when they transmit through the network.
- HMAC can be used. It provides authentication by hashing the message itself. At the end the output is hashed again with the key.

7. Conclusion

This document describes the extension of the key length 128 bit to 192 and 256-bits. These key lengths are used in both encryption and decryption. Based on the length of the key the round values will be increased. AES is an efficient algorithm which uses iterative approach unlike DES that uses fiestel cipher structure. The best authentication strategy we would like to suggest is SHA-3.It is used in many algorithms to provide authentication.

8. References

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf
https://www.rivier.edu/journal/ROAJ-Fall-2010/J455-Selent-AES.pdf

9. Appendices