ASSIGNMENT - 1

Network and System Security(SIL-765)

ADVANCED ENCRYPTION STANDARD

RUCHIR DHIMAN

(2016MCS2685)

SHANTANU AGARWAL

(2016MCS2661)

Introduction

The Advanced Encryption Standard, or AES, is a symmetric block cipher chosen by the U.S. government to protect classified information and is implemented in software and hardware throughout the world to encrypt sensitive data.

The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for a successor algorithm for the Data Encryption Standard (DES), which was starting to become vulnerable to brute-force attacks.

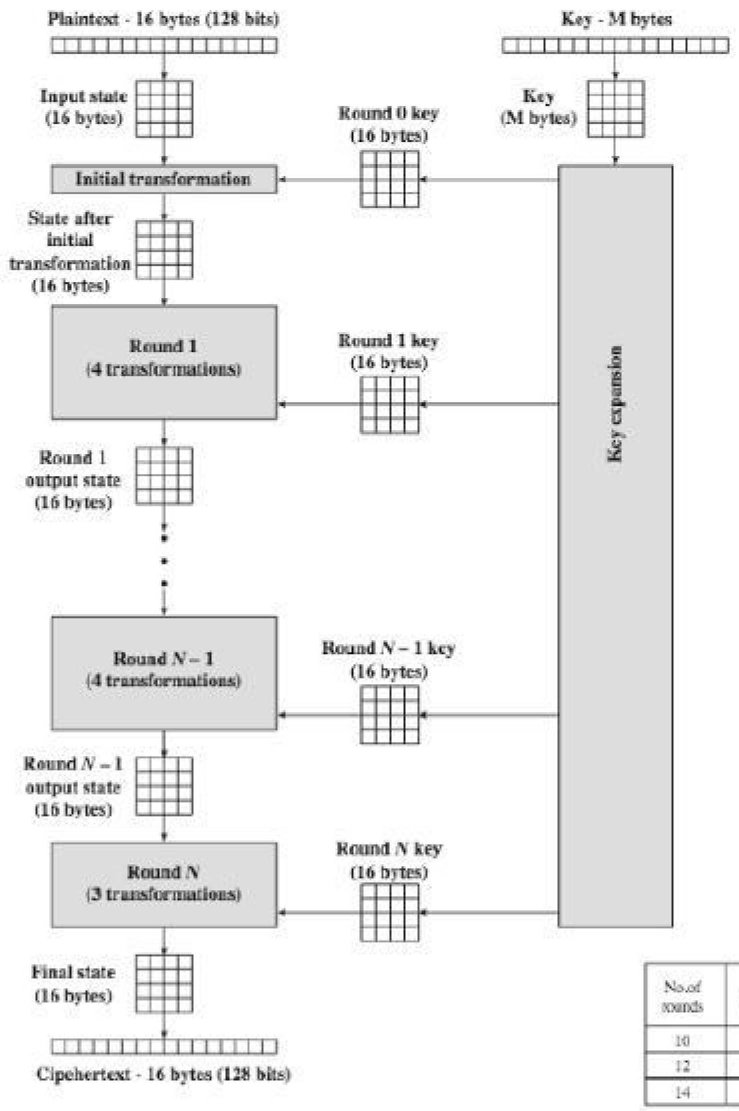
This new, advanced encryption algorithm would be unclassified and had to be "capable of protecting sensitive government information well into the next century," according to the NIST announcement of the process for development of an advanced encryption standard algorithm. It was intended to be easy to implement in hardware and software, as well as in restricted environments (for example, in a smart card) and offer good defenses against various attack techniques.

AES Structure

The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix.

Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words. Figure 5.2b shows the expansion for the 128-bit key. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that the ordering of bytes within a matrix is by column.


Plaintext - 16 bytes (128 bits)
Key - M bytes
Input state (16 bytes)
Round 0 key (16 bytes)
Key (M bytes)
Initial transformation
State after initial transformation (16 bytes)
Round 1 (4 transformations)
Round 1 key (16 bytes)
Key expansion
Round 1 output state (16 bytes)
Round N – 1 (4 transformations)
Round N – 1 key (16 bytes)
Round N – 1 output state (16 bytes)
Round N (3 transformations)
Round N key (16 bytes)
Final state (16 bytes)
Cipehertext - 16 bytes (128 bits)

No.of rounds	Key Length (bytes)
10	16
12	24
14	32

Implementation

We implemented the AES algorithm in JAVA using 'eclipse IDE'. No inbuilt function is used, everything is built from scratch.

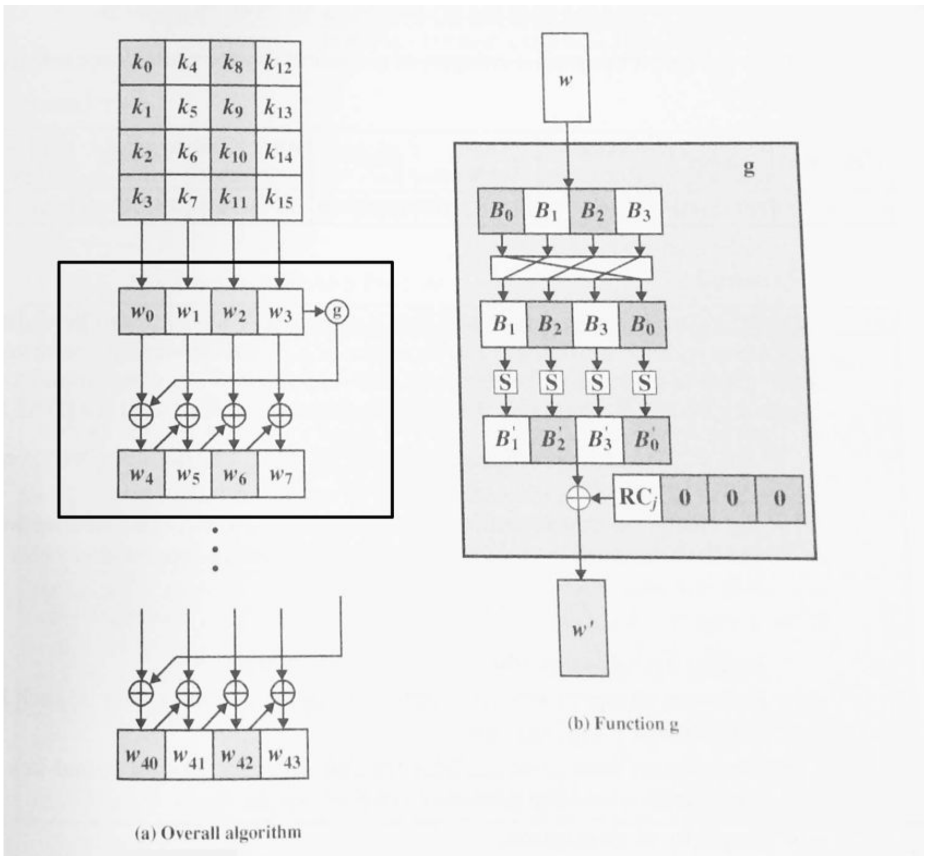
We have implemented following steps in building this system:

STEP 1: AES Key expansion

We took input as 4 words, where each word is 4 bytes long and we generated the 44 words out of the 4 words. The key is copied to the first four words of the expanded key without any modification. Rest any word w[i] is generated using following logic:

W[i] = W[i-4] XOR g(W[i-1]) if 'i' is a multiple of 4,

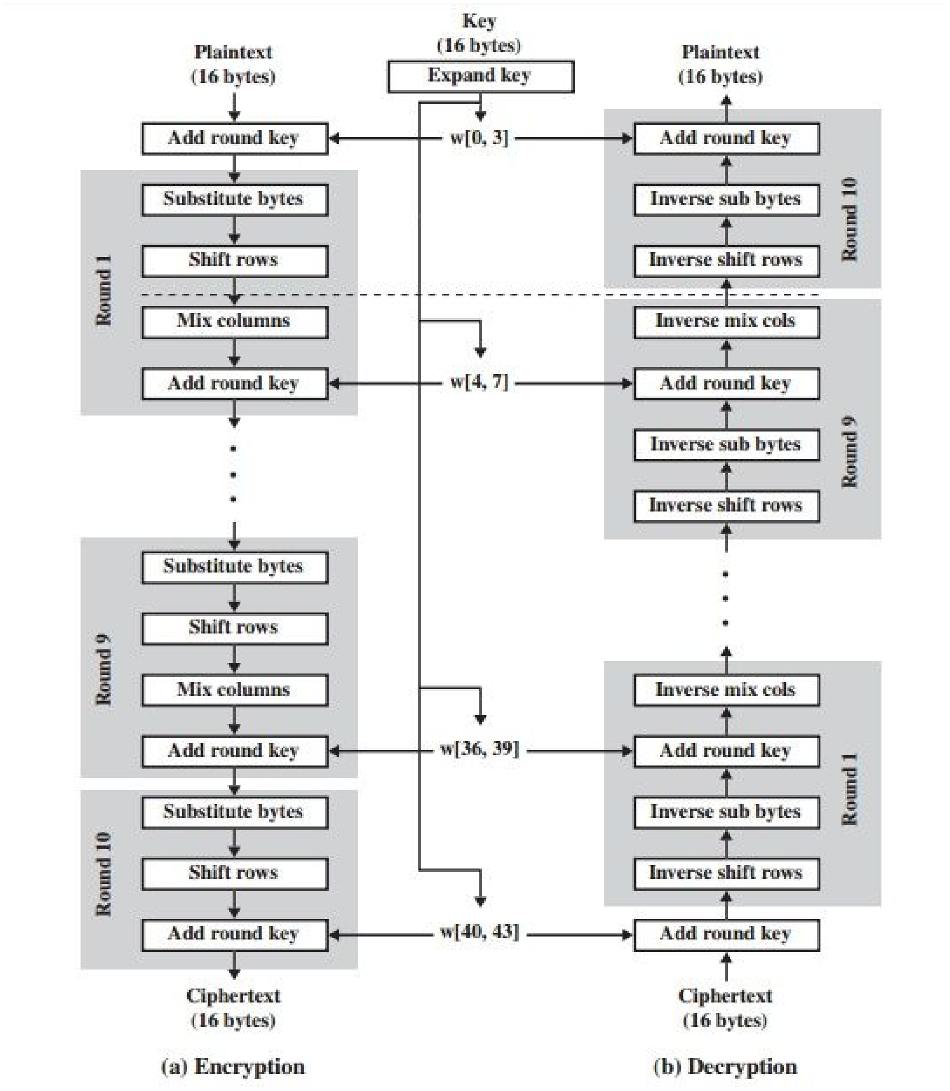
W[i] = W[i-4] XOR W[i-1] otherwise


k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7
w_{40} w_{41} w_{42} w_{43}
(a) Overall algorithm
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B'_1 B'_2 B'_3 B'_0
RC_j 0 0 0
w'
(b) Function g

AES Key Expansion

STEP 2: AES Encryption and Decryption

The algorithm for encryption and decryption in AES looks like:



Encryption starts with an AddRoundKey function , after that, every round consists of four different stages, one for permutation and three for substitution.

All the functions are easily reversible without the knowledge of the key except the AddRoundKey. And AddRoundKey is nothing but XOR operation of output with the key. Thus the decryption is easy reversible process of encryption.

Four stages that are implemented are as follows:

1. Substitute bytes: Uses an S-box to perform byte-by-byte substitution of the block.
2. Shift Rows: A simple permutation.
3. MixColumns: A substitution that makes use of arithmetic over GF(28) .
4. AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key.

Implementation Detail

- No library functions are used to develop the system.
- Language used is: JAVA

JAVA Classes

Driver.java:

This class is used to take input from user and perform preprocessing before the actual algorithm.

Helpers.java

This class contain some basic function for string and bit manipulation. These functions are used by all other classes including Encryption class, decryption class and Keys class. Some basic functions include, S-box and inverse S-box, character to binary, binary to string etc.

Keys.java

This class contains the functions necessary to generate the keys for all the rounds. It contains a array of keys. Keys are generated before encryption or decryption and then these keys are being used for the same.

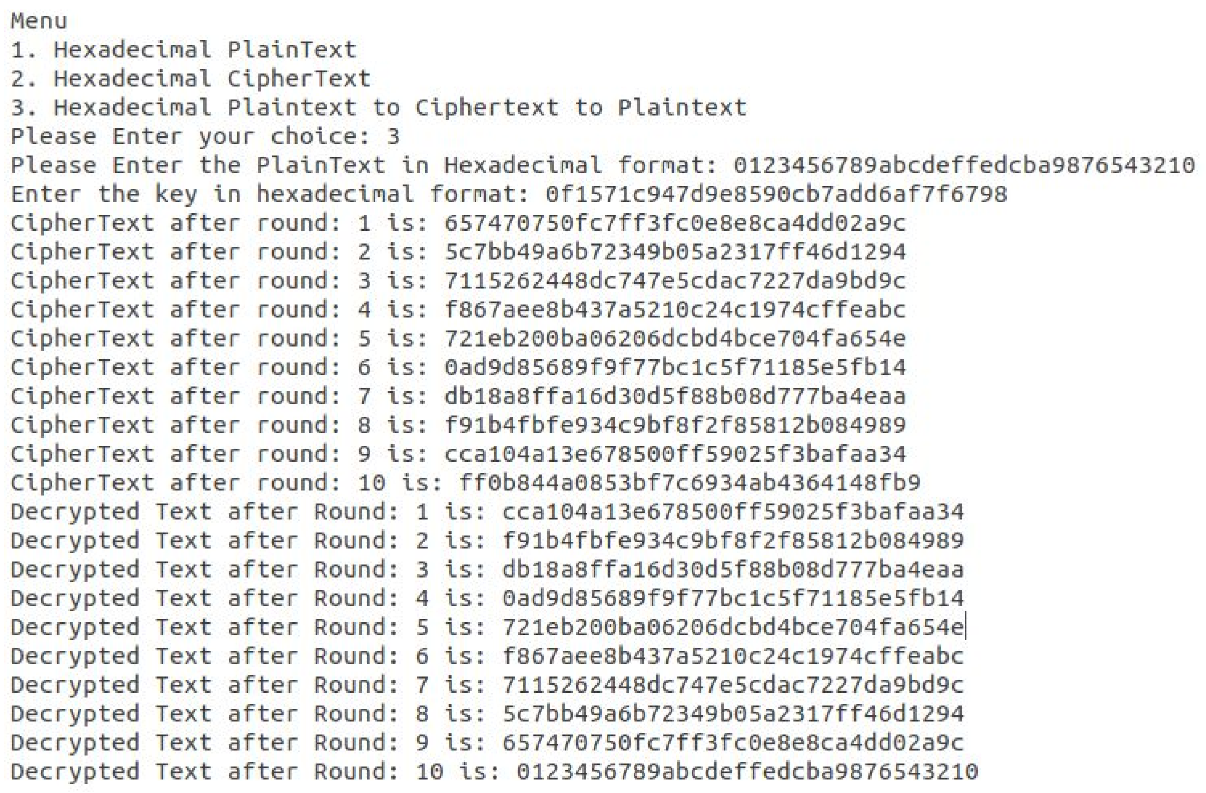
Encryptor.java

This class contains the basic functionality for AES plainText encryption to convert the same into the ciphertext.

Decryptor.java

This class contains the basic functionality for AES cipherText decryption to obtain the plainText.

Sample Run



```
Menu
1. Hexadecimal PlainText
2. Hexadecimal CipherText
3. Hexadecimal Plaintext to Ciphertext to Plaintext
Please Enter your choice: 3
Please Enter the PlainText in Hexadecimal format: 0123456789abcdeffedcba9876543210
Enter the key in hexadecimal format: 0f1571c947d9e8590cb7add6af7f6798
CipherText after round: 1 is: 657470750fc7ff3fc0e8e8ca4dd02a9c
CipherText after round: 2 is: 5c7bb49a6b72349b05a2317ff46d1294
CipherText after round: 3 is: 7115262448dc747e5cdac7227da9bd9c
CipherText after round: 4 is: f867aee8b437a5210c24c1974cffeabc
CipherText after round: 5 is: 721eb200ba06206dcbd4bce704fa654e
CipherText after round: 6 is: 0ad9d85689f9f77bc1c5f71185e5fb14
CipherText after round: 7 is: db18a8ffa16d30d5f88b08d777ba4eaa
CipherText after round: 8 is: f91b4fbfe934c9bf8f2f85812b084989
CipherText after round: 9 is: cca104a13e678500ff59025f3bafaa34
CipherText after round: 10 is: ff0b844a0853bf7c6934ab4364148fb9
Decrypted Text after Round: 1 is: cca104a13e678500ff59025f3bafaa34
Decrypted Text after Round: 2 is: f91b4fbfe934c9bf8f2f85812b084989
Decrypted Text after Round: 3 is: db18a8ffa16d30d5f88b08d777ba4eaa
Decrypted Text after Round: 4 is: 0ad9d85689f9f77bc1c5f71185e5fb14
Decrypted Text after Round: 5 is: 721eb200ba06206dcbd4bce704fa654e
Decrypted Text after Round: 6 is: f867aee8b437a5210c24c1974cffeabc
Decrypted Text after Round: 7 is: 7115262448dc747e5cdac7227da9bd9c
Decrypted Text after Round: 8 is: 5c7bb49a6b72349b05a2317ff46d1294
Decrypted Text after Round: 9 is: 657470750fc7ff3fc0e8e8ca4dd02a9c
Decrypted Text after Round: 10 is: 0123456789abcdeffedcba9876543210
```

As can be seen, the results are identical for round 'X' of encryption to the result of round 'N-X' of decryption.