Assignment No:

Title of the Assignment: AES Algorithm

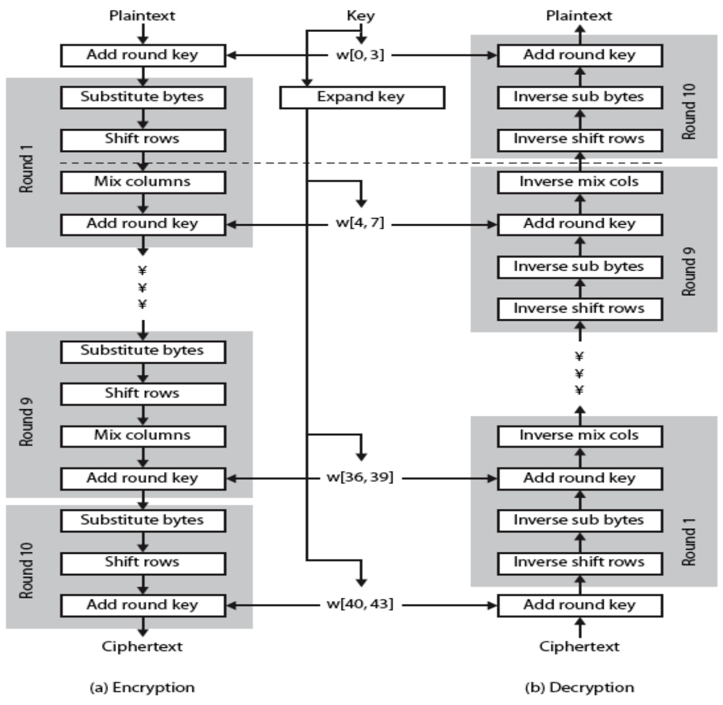
Problem statement: Write a Java/C/C++/Python program to implement AES Algorithm.

Objective:

- To understand the encryption/decryption using AES.
- To implement AES using JAVA.

Theory:

AES is a symmetric block cipher. This means that it uses the same key. A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively. The input is a single 128 bit block both for decryption and encryption and is known as the **in** matrix. This block is copied into a **state** array which is modified at each stage of the algorithm and then copied to an output matrix. Both the plaintext and key are depicted as a 128 bit square matrix of bytes. This key is then expanded into an array of key schedule words (the **w** matrix). The overall structure of AES is as below.



The algorithm begins with an **Add round key** stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of it's counterpart in the encryption algorithm. The four stages are as follows:

1. Substitute bytes
2. Shift rows
3. Mix Columns
4. Add Round Key

The tenth round simply leaves out the **Mix Columns** stage.

1. Substitute Bytes

This stage (known as SubBytes) is simply a table lookup using a 16×16 matrix of byte values called an **s-box**. This matrix consists of all the possible combinations of an 8 bit sequence (28 = 16 × 16 = 256). Again the matrix that gets operated upon throughout the encryption is known as **state**. We will be concerned with how this matrix is effected in each round. For this particular round each byte is mapped into a new byte in the following way: the leftmost nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column. For example, the byte {95} (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value {2A}. This is then used to update the **state** matrix. Below figure shows this idea.

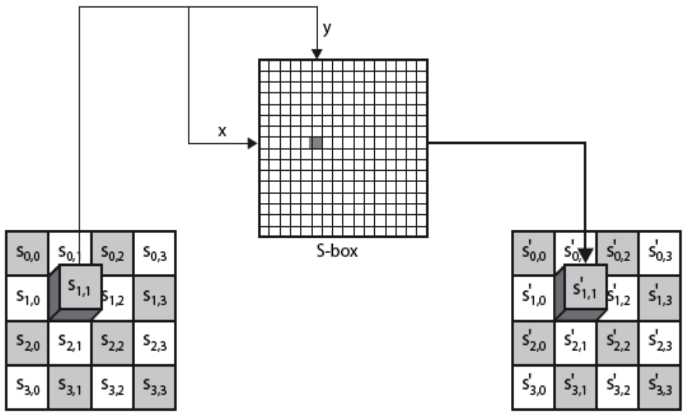
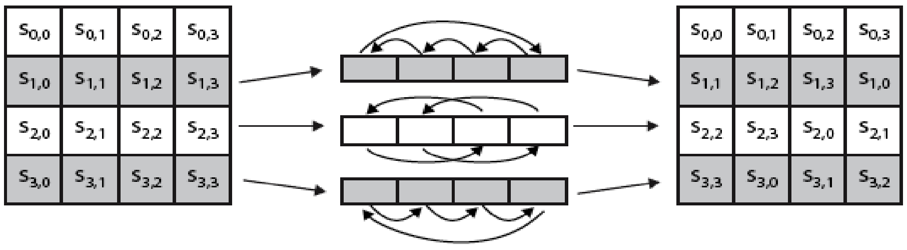

y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Fig. Substitute Byte Stage of AES

2. Shift Row Transformation

This stage (known as ShiftRows) is shown in figure below. This is a simple permutation and nothing more. It works as follow:

- The first row of **state** is *not* altered.
- The second row is shifted 1 bytes to the left in a circular manner.
- The third row is shifted 2 bytes to the left in a circular manner.
- The fourth row is shifted 3 bytes to the left in a circular manner.



$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

Fig: Shift Row Transformation

3. Mix Column Transformation

This stage (known as MixColumn) is basically a substitution but it makes use of arithmetic of GF(28). Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. The transformation can be determined by the following matrix multiplication on **state.**

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Each element of the product matrix is the sum of products of elements of one row and one column. In this case the individual additions and multiplications are performed in GF(28). The MixColumns transformation of a single column j (0 _ j _ 3) of **state** can be expressed as:

$$
\begin{aligned}
s'_{0,j} &= (2 \bullet s_{0,j}) \oplus (3 \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\
s'_{1,j} &= s_{0,j} \oplus (2 \bullet s_{1,j}) \oplus (3 \bullet s_{2,j}) \oplus s_{3,j} \\
s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \bullet s_{2,j}) \oplus (3 \bullet s_{3,j}) \\
s'_{3,j} &= (3 \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \bullet s_{3,j})
\end{aligned}
$$

where • denotes multiplication over the finite field GF(2^8).

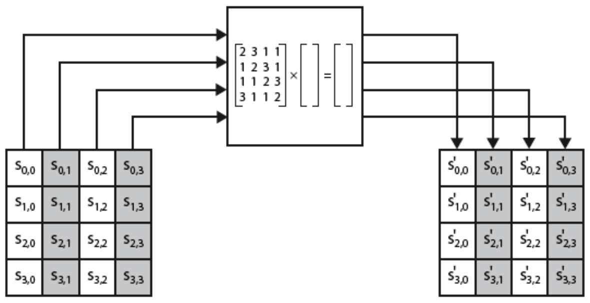


Fig: Mix Column Stage

4. Add Round Key Transformation

In this stage (known as AddRoundKey) the 128 bits of **state** are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column wise operation between the 4 bytes of a **state** column and one word of the round key. This transformation is as simple as possible which helps in efficiency but it also affects every bit of **state**.

4.1 AES Key Expansion

The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure 7.2. Each word contains 32 bytes which means each subkey is 128 bits long. Below figure illustrates the generation of the first eight words of the expanded key using the symbol g to represent that complex function.

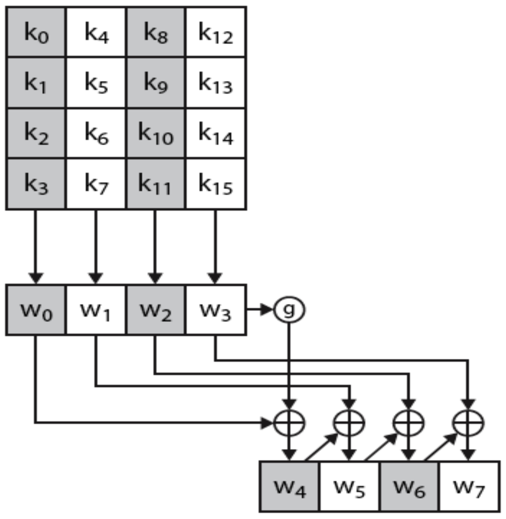


Fig: AES Key Expansion

The function g consists of the following subfunctions:

1. **RotWord** performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. **SubWord** performs a byte substitution on each byte of its input word, using the s-box described earlier.
3. The result of steps 1 and 2 is XORed with round constant, Rcon[j].

Pseudo code for AES:

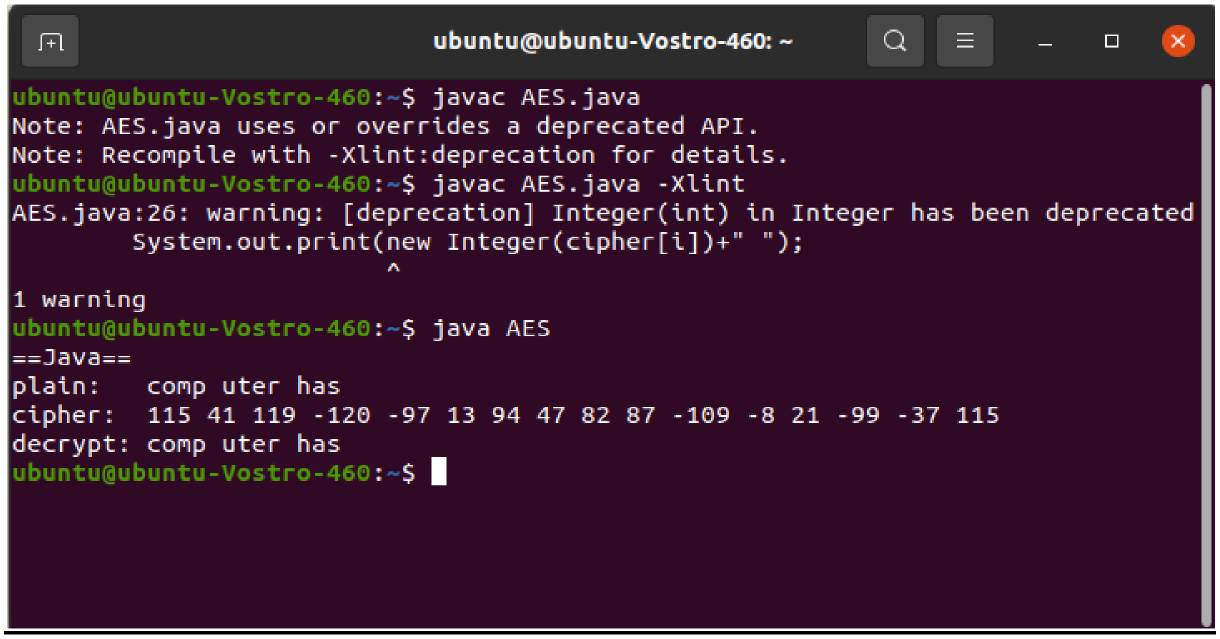
```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
byte state[4,Nb]
state = in
AddRoundKey(state, w[0, Nb-1])
for round = 1 step 1 to Nr–1
     SubBytes(state)
     ShiftRows(state)
     MixColumns(state)
     AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for
SubBytes(state)
```

```
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
out = state
end
```

Program:

```
import java.security.MessageDigest;
import java.util.Arrays;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.spec.IvParameterSpec;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class AES {
  static String IV = "AAAAAAAAAAAAAAAA";
  static String plaintext = "comp uter has\u0000\u0000\u0000"; /*Note null 
padding*/
  static String encryptionKey = "0123456789abcdef";
  public static void main(String [] args) {
    try {

      System.out.println("==Java==");
      System.out.println("plain:   " + plaintext);

      byte[] cipher = encrypt(plaintext, encryptionKey);

      System.out.print("cipher:  ");
      for (int i=0; i<cipher.length; i++)
        System.out.print(new Integer(cipher[i])+" ");
      System.out.println("");

      String decrypted = decrypt(cipher, encryptionKey);

      System.out.println("decrypt: " + decrypted);

    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static byte[] encrypt(String plainText, String encryptionKey) 
throws Exception {
    Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding", "SunJCE");
    SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"),
"AES");
    cipher.init(Cipher.ENCRYPT_MODE, key,new
IvParameterSpec(IV.getBytes("UTF-8")));
    return cipher.doFinal(plainText.getBytes("UTF-8"));
  }

  public static String decrypt(byte[] cipherText, String encryptionKey)
throws Exception{
    Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding", "SunJCE");
    SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"),
"AES");
    cipher.init(Cipher.DECRYPT_MODE, key,new
IvParameterSpec(IV.getBytes("UTF-8")));
    return new String(cipher.doFinal(cipherText),"UTF-8");
  }
}
```

Output:



```
ubuntu@ubuntu-Vostro-460:~$ javac AES.java
Note: AES.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
ubuntu@ubuntu-Vostro-460:~$ javac AES.java -Xlint
AES.java:26: warning: [deprecation] Integer(int) in Integer has been deprecated
        System.out.print(new Integer(cipher[i])+" ");
                         ^
1 warning
ubuntu@ubuntu-Vostro-460:~$ java AES
==Java==
plain:   comp uter has
cipher:  115 41 119 -120 -97 13 94 47 82 87 -109 -8 21 -99 -37 115
decrypt: comp uter has
ubuntu@ubuntu-Vostro-460:~$ 
```

Conclusion:

We have implemented AES algorithm using Java.

Oral questions:

1. What is power analysis?
2. What was the final set of criteria used by NIST to evaluate candidate AES ciphers?

3. What is the purpose of the **State** array?
4. How is s-box constructed?
5. Briefly describe SubBytes and ShiftRows.
6. Briefly describe the key expansion algorithm.
7. What is the difference between ShiftRows and RotWord?
8. List the parameters (blocksize, key size, and the number of rounds) for the three AES versions.
9. Why do you think the mixing transformation (MixColumns) is not needed in Des, but is needed in AES?
10. Are the s-boxes (substitution tables) in AEs static or dynamic?
11. In AES-128, the round key used in the pre-round operation is the same as the cipher key. Is this the case for AEs-192 and AES-256?