PAGE NO.: 12
DATE : / / 20

Practical No. 6

Aim:- To write a program to implement AES algorithm.

Theory:- AES stands for Advanced Encryption standard. AES comes under block cipher symmetric key cryptography. AES is widely used today as it is a much stronger that DES and despite being hard to implement. The key size can be 128/192/256 bits.

Working of the cipher ⇒

AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits of the input data at a time. The number of rounds depends on the key length as follows ⇒

* 128 bit key – 10 rounds
* 132 bit key – 12 rounds
* 256 bit key – 14 rounds.

Algorithm ⇒ The algorithm will work as follows :-

1) Creation of round keys ⇒ A key schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys.

2) Encryption ⇒ AES considers each block as a 16 byte (4x4) grid in a column major arrangement.

b_0	b_4	b_8	b_{12}
b_1	b_5	b_9	b_{13}
b_2	b_6	b_{10}	b_{14}
b_3	b_7	b_{11}	b_{15}

Each round comprises of 4 steps ⇒

- SubBytes: Each byte is substituted by another byte. It is performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte.

Teacher's Signature _____

- Shift rows: Each row is shifted a particular number of times
  - The first row is not shifted.
  - The second row is shifted once to the left.
  - The third row is shifted twice to the left.
  - The fourth row is shifted thrice to the left.

$$\begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 & b_7 \\ b_8 & b_9 & b_{10} & b_{11} \\ b_{12} & b_{13} & b_{14} & b_{15} \end{bmatrix} \rightarrow \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_5 & b_6 & b_7 & b_4 \\ b_{10} & b_{11} & b_8 & b_9 \\ b_{15} & b_{12} & b_{13} & b_{14} \end{bmatrix}$$

- Mix columns: It is basically a matrix multiplication, each column is multiplied with a specific matrix and thus the column changed.

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- Add round keys:- Now the resultant output of the previous stage is XOR-ed with the corresponding round key.
- Decryption:- The stages in the rounds can be easily undone as these stages have an opposite to it with when performed reverts changes. Stages of each rounds in decryption is as follows:-
- Inverse Mix Columns: It is similar to the mix columns steps in encryption, but differs in matrix used to carry out the operation.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

- Inverse subBytes:- Inverse S-box is used as a lookup table and is using which the bytes are substituted during decryption.

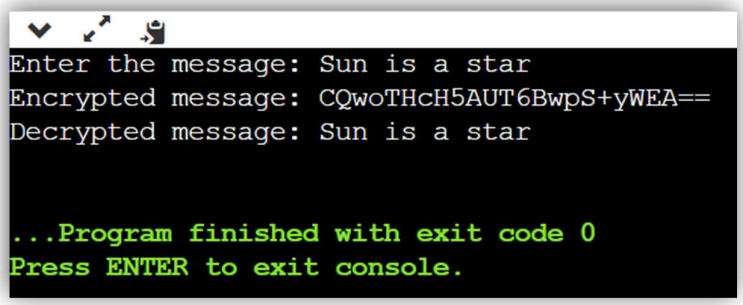
Ashakaur

Program:

```
import java.nio.charset.StandardCharsets;
import java.security.spec.KeySpec;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.*;

class AES {
	private static final String SECRET_KEY = "my_super_secret_key_ho_ho_ho";
	private static final String SALT = "ssshhhhhhhhhhh!!!!";
	public static String encrypt(String strToEncrypt){
		try {
			byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
			IvParameterSpec ivspec = new IvParameterSpec(iv);
			SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
			KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(),
SALT.getBytes(), 65536, 256);
			SecretKey tmp = factory.generateSecret(spec);
			SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
			Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
			cipher.init(Cipher.ENCRYPT_MODE, secretKey,ivspec);
			return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_
8)));
		}
		catch (Exception e) {
			System.out.println("Error while encrypting: "+ e.toString());
		}
		return null;
	}

	public static String decrypt(String strToDecrypt){
		try {
			byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
			IvParameterSpec ivspec = new IvParameterSpec(iv);
			SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
			KeySpec spec = new PBEKeySpec(SECRET_KEY.toCharArray(),
SALT.getBytes(), 65536, 256);
			SecretKey tmp = factory.generateSecret(spec);
			SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
			Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
			cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);
			return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
		}
		catch (Exception e) {
```

```
                System.out.println("Error while decrypting: " + e.toString());
            }
            return null;
        }
}

public class Main {
        public static void main(String[] args){
           Scanner sc = new Scanner(System.in);
                System.out.print("Enter the message: ");
                String originalString = sc.nextLine();
                String encryptedString = AES.encrypt(originalString);
                String decryptedString = AES.decrypt(encryptedString);
                System.out.println("Encrypted message: " + encryptedString);
                System.out.println("Decrypted message: " + decryptedString);
        }
}
```

Output:



Conclusion:

The program to implement AES algorithm has been executed successfully.