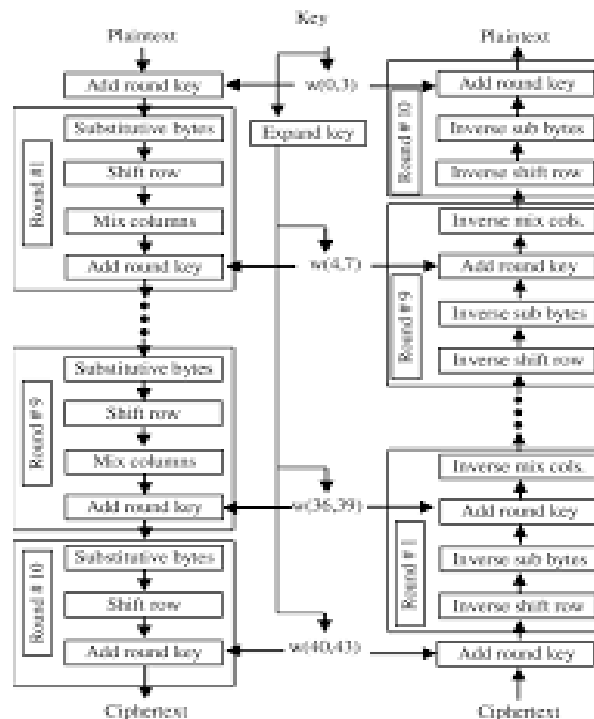


### AES Algorithm:-

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. AES can use a block length of 128 bits and a key length that can be 128, 192 or 256 bits. In description of this section, we assume a key length of 128 bits. This is likely to be one most commonly implemented.



The encryption phase of AES can be broken into three phases: the initial round, the main rounds, and the final round. All of the phases use the same sub-operations in different combinations as follows:

Initial Round

AddRoundKey

Main Rounds

SubBytes

ShiftRows

MixColumns

AddRoundKey

Final Round

SubBytes

ShiftRows

AddRoundKey

The main rounds of AES are repeated a set number of times for each variant of AES. AES-128 uses 9 iterations of the main round, AES-192 uses 11, and AES-256 uses 13

AddRoundKey:-

The AddRoundKey operation is the only phase of AES encryption that directly operates on the AES round key. In this operation, the input to the round is exclusive-ored with the round key.

SubBytes:-

The SubBytes phase of AES involves splitting the input into bytes and passing each through a Substitution Box or S-Box. Unlike DES, AES uses the same S-Box for all bytes. The AES S-Box implements inverse multiplication in Galois Field 28. The AES S-Box is shown in the Table below.

ShiftRows

In the ShiftRows phase of AES, each row of the 128-bit internal state of the cipher is shifted. The rows in this stage refer to the standard representation of the internal state in AES, which is a 4x4 matrix where each cell contains a byte. Bytes of the internal state are placed in the matrix across rows from left to right and down columns. In the ShiftRows operation, each of these rows is shifted to the left by a set amount: their row number starting with zero.

MixColumns

Like the ShiftRows phase of AES, the MixColumns phase provides diffusion by mixing the input around. Unlike ShiftRows, MixColumns performs operations splitting the matrix by columns instead of rows.

A visual representation of the MixColumns operation is shown above. Unlike standard matrix multiplication, MixColumns performs matrix multiplication as per Galois Field 28. Although we won't describe this step in detail, it is important to note that this multiplication has the property of operating independently over each of the columns of the initial matrix, i.e. the first column when multiplied by the matrix, produces the first column of the resultant matrix.

he AES Key Schedule

The AES Key Schedule is used to produce a set number of round keys from the initial key. In AES, the initial key is used in the initial round of AES as input to the AddRoundKey operation. From this key, 10, 12, or 14 round keys are produced as input to the other AddRoundKey operations in the 128, 192, and 256-bit versions of AES.

### Decryption in AES

To decrypt an AES-encrypted ciphertext, it is necessary to undo each stage of the encryption operation in the reverse order in which they were applied. The three stage of decryption are as follows:

#### Inverse Final Round

AddRoundKey

ShiftRows

SubBytes

#### Inverse Main Round

AddRoundKey

MixColumns

ShiftRows

SubBytes

#### Inverse Initial Round

AddRoundKey

Of the four operations in AES encryption, only the AddRoundKey operation is its own inverse (since it is an exclusive-or). To undo AddRoundKey, it is only necessary to expand the entire AES key schedule (identically to encryption) and then use the appropriate key in the exclusive-or. The other three operations require an inverse operation to be defined and used. The first operation to be undone is ShiftRows. The Inverse ShiftRows operation is identical to the ShiftRows operation except that rotations are made to the right instead of to the left. This is illustrated in the Figure below.

The reason why this multiplication inverts the initial operation is because of how math works in the Galois Field 28, which we won't describe in detail in this tutorial. However, note that the specific values in both matrices are chosen in a way such that one multiplication is the inverse of the other in Galois Field 28. AES ciphertexts are decrypted by following the order of operations explained at the beginning of this section, using the appropriate inverse operations and using round keys in reverse order.

Program:-

```
import javax.swing.*.*;
```

```
import java.security.SecureRandom;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Random ;

class AES {

    byte[] skey = new byte[1000];

    String skeyString;

    static byte[] raw;

    String inputMessage,encryptedData,decryptedMessage;

    public AES() {

        try {

            generateSymmetricKey();

            inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");

            byte[] ibyte = inputMessage.getBytes();

            byte[] ebyte=encrypt(raw, ibyte);

            String encryptedData = new String(ebyte);

            System.out.println("Encrypted message "+encryptedData);

            JOptionPane.showMessageDialog(null,"Encrypted Data "+"\\n"+encryptedData);

            byte[] dbyte= decrypt(raw,ebyte);

            String decryptedMessage = new String(dbyte);

            System.out.println("Decrypted message "+decryptedMessage);

            JOptionPane.showMessageDialog(null,"Decrypted Data "+"\\n"+decryptedMessage);

        }

    }

}
```

```
catch(Exception e) {  
    System.out.println(e);  
}  
}  
  
void generateSymmetricKey() {  
    try {  
        Random r = new Random();  
        int num = r.nextInt(10000);  
        String knum = String.valueOf(num);  
        byte[] knumb = knum.getBytes();  
        skey=getRawKey(knumb);  
        skeyString = new String(skey);  
        System.out.println("AES Symmetric key = "+skeyString);  
    }  
    catch(Exception e) {  
        System.out.println(e);  
    }  
}  
  
private static byte[] getRawKey(byte[] seed) throws Exception {  
    KeyGenerator kgen = KeyGenerator.getInstance("AES");  
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");  
    sr.setSeed(seed);  
    kgen.init(128, sr); // 192 and 256 bits may not be available  
    SecretKey skey = kgen.generateKey();  
    raw = skey.getEncoded();  
}
```

```
return raw;

}

private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {

    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

    Cipher cipher = Cipher.getInstance("AES");

    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

    byte[] encrypted = cipher.doFinal(clear);

    return encrypted;

}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {

    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

    Cipher cipher = Cipher.getInstance("AES");

    cipher.init(Cipher.DECRYPT_MODE, skeySpec);

    byte[] decrypted = cipher.doFinal(encrypted);

    return decrypted;

}

public static void main(String args[]) {

    AES aes = new AES();

}

}
```

O/P:-

Command Prompt - java AES

```
encrypted-text=f5f44f5f0e67784d -> decrypted-text(source text)=03215672
```

```
D:\>javac AES.java
```

```
D:\>java AES
```

```
AES Symmetric key = '?+ç?V?Ü ??hî?<`
```

```
Encrypted message ·?ö4??çIX»?G%
```

```
Decrypted message YASH12345
```

```
D:\>javac DesedeCrypter.java
```

```
D:\>java DesedeCrypter
```

```
sourceText=03215672 -> encryptedText=f5f44f5f0e67784d
```

```
encrypted-text=f5f44f5f0e67784d -> decrypted-text(source text)=03215672
```

```
D:\>javac AES.java
```

```
D:\>javac DesedeCrypter.jav
```

```
error: Class names, 'DesedeCrypter.jav', are only accepted if annotation processing is exp  
1 error
```

```
D:\>
```

```
D:\>javac AES
```

```
error: Class names, 'AES', are only accepted if annotation processing is explicitly reques  
1 error
```

```
D:\>javac AES.java
```

```
D:\>javac AES
```

```
error: Class names, 'AES', are only accepted if annotation processing is explicitly reques  
1 error
```

```
D:\>javac AES.java
```

```
D:\>java AES
```

```
AES Symmetric key = ft?%#????=iI?T
```

```
D:\>
```

```
D:\>
```

```
D:\>java AES
```

```
AES Symmetric key = ???v?¥??úr%?dEZ?
```



2)

Command Prompt - java AES

D:\>javac AES.java

D:\>java AES

AES Symmetric key = '?+ç?V?Ü ??hî?<`

Encrypted message ·?@?ö4@??çIX»?G%

Decrypted message YASH12345

D:\>javac DesedeCrypter.java

D:\>java DesedeCrypter

sourceText=03215672 -> encryptedText=f5f44f5f0e67784d

encrypted-text=f5f44f5f0e67784d -> decrypted-text(source text)=03215672

D:\>javac AES.java

D:\>javac DesedeCrypter.jav

error: Class names, 'DesedeCrypter.jav', are only accepted if annotation processing is enabled  
1 error

D:\>

D:\>javac AES

error: Class names, 'AES', are only accepted if annotation processing is explicitly requested  
1 error

D:\>javac AES.java

D:\>javac AES

error: Class names, 'AES', are only accepted if annotation processing is explicitly requested  
1 error

D:\>javac AES.java

D:\>java AES

AES Symmetric key = ft@?%@#????=iI?T

D:\>

D:\>

D:\>java AES

AES Symmetric key = ???v?¥??úr%?dEZ?

D:\>java AES

AES Symmetric key = \_Suº E?W@ A6@%??

Input

Enter message to encrypt

AES

OK Cancel

3)



```

C:\> Command Prompt - java AES
Encrypted message ·?040??çIX»?G%
Decrypted message YASH12345

D:\>javac DesedeCrypter.java

D:\>java DesedeCrypter
sourceText=03215672 -> encryptedText=f5f44f5f0e67784d

encrypted-text=f5f44f5f0e67784d -> decrypted-text(source text)=03215672

D:\>javac AES.java

D:\>javac DesedeCrypter.jav
error: Class names, 'DesedeCrypter.jav', are only accepted if annotation processing is exp
1 error

D:\>
D:\>javac AES
error: Class names, 'AES', are only accepted if annotation processing
1 error

D:\>javac AES.java

D:\>javac AES
error: Class names, 'AES', are only accepted if annotation processing
1 error

D:\>javac AES.java

D:\>java AES
AES Symmetric key = ft0?%0#????=iI?T

D:\>
D:\>
D:\>java AES
AES Symmetric key = ???v?¥??úr%?dEZ?

D:\>java AES
AES Symmetric key = _Su°          E?W0 A60%??

D:\>java AES
???050mmetric key = ùPè0è0?'
Encrypted message !0??î0U[?Ñd;~05V

```



3)

Command Prompt

```
encrypted-text=f5f44f5f0e67784d -> decrypted-text(source text)=03215672
```

```
D:\>javac AES.java
```

```
D:\>javac DesedeCrypter.jav
```

```
error: Class names, 'DesedeCrypter.jav', are only accepted if annotation processing is exp  
1 error
```

```
D:\>
```

```
D:\>javac AES
```

```
error: Class names, 'AES', are only accepted if annotation processing is explicitly reques  
1 error
```

```
D:\>javac AES.java
```

```
D:\>javac AES
```

```
error: Class names, 'AES', are only accepted if annotation processing is explicitly reques  
1 error
```

```
D:\>javac AES.java
```

```
D:\>java AES
```

```
AES Symmetric key = ft0?%0#????=iI?T
```

```
D:\>
```

```
D:\>
```

```
D:\>java AES
```

```
AES Symmetric key = ???v?¥??úr%?dEZ?
```

```
D:\>java AES
```

```
AES Symmetric key = _Su0 E?W0 A60%??
```

```
D:\>java AES
```

```
???050mmetric key = ûPè0è0?'
```

```
Encrypted message !0??î0U[?Ñd;~05V
```

```
D:\>java AES
```

```
:ES Symmetric key = ín?%?016Bf????
```

```
Encrypted message ýû??0?
```

```
0;0'0L~ß?
```

```
Decrypted message AES
```

```
D:\>
```



RSA Algorithm:-

One of the first public-key schemes was developed in 1977 . The RSA scheme has since that time reigned supreme as the most widely accepted and implemented approach public-key encryption . RSA is a block cipher in which the plain text and ciphertext are integer between 0 and  $n-1$  for some  $n$ .

Algorithm :-

Key Generation	
Select $p, q$	$P$ and $q$ are prime , $p \neq q$
Calculate $n = p * q$	
Calculate $\Theta(n) = (p-1)(q-1)$	
Select integer $e$	$\text{Gcd}(\Theta(n), e) = 1; 1 < e < \Theta(n)$
Calculate $d$	$de \bmod \Theta(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext :  $M < n$

Ciphertext  $C = M^e \pmod{n}$

Decryption

Plaintext :  $C$

Ciphertext  $M = C^d \pmod{n}$

Program:-

```
import java.util.*;
```

```
import java.math.*;
```

```
class RSA{
```

```
    public static void main(String args[])
```

```
{
```

```
    Scanner sc=new Scanner(System.in);
```

```
    int p,q,n,z,d=0,e,i;
```

```
    System.out.println("Enter the number to be encrypted and decrypted");
```

```
int msg=sc.nextInt();

double c;

BigInteger msgback;

System.out.println("Enter 1st prime number p");

p=sc.nextInt();

System.out.println("Enter 2nd prime number q");

q=sc.nextInt();

n=p*q;

z = (p-1)*(q-1);

System.out.println("the value of z = "+z);

for(e=2;e<z;e++)

{

    if(gcd(e,z)==1)        // e is for public key exponent

    {

        break;

    }

}

System.out.println("the value of e = "+e);
```

```
for(i=0;i<=9;i++)

{

    int x=1+(i*z);

    if(x%e==0)    //d is for private key exponent

    {

        d=x/e;

        break;

    }

}

System.out.println("the value of d = "+d);

c=(Math.pow(msg,e))%n;

System.out.println("Encrypted message is : -");

System.out.println(c);

BigInteger N = BigInteger.valueOf(n);

BigInteger C = BigDecimal.valueOf(c).toBigInteger();

msgback = (C.pow(d)).mod(N);

System.out.println("Derypted message is : -");

System.out.println(msgback);
```

```
}

static int gcd(int e, int z)

{

    if(e==0)

        return z;


    else

        return gcd(z%e,e);

}

}
```

O/P:-

 Command Prompt

```
12
Enter 1st prime number p
3
Enter 2nd prime number q
11
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is : -
12.0
Derypted message is : -
12

D:\>java RSA
Enter the number to be encrypted and decrypted
456
Enter 1st prime number p
3
Enter 2nd prime number q
11
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is : -
15.0
Derypted message is : -
27

D:\>java RSA
Enter the number to be encrypted and decrypted
12
Enter 1st prime number p
3
Enter 2nd prime number q
11
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is : -
12.0
Derypted message is : -
12

D:\>
```



Type here to search



### 3DES algorithm:-

3DES was first standardized for use in functional applications in ANSI standard X9.17 in 1985. 3DES was incorporated as part of the data encryption standard in 1999 in with 1999 with the public encryption of FIPS 46-3. 3DES uses three keys and three execution of the DES algorithm . The function follows an encrypt-decrypt-encrypt(EDE) sequence.

$$C = E(K_3, D(K_2, E(K_1, P)))$$

C=ciphertext

P=plain text

E[K,X]=encryption of X using key K

D[K,Y]=decryption of Y using key K

### Decryption algorithm:-

$$C = E(K_1, E(K_2, E(K_1, P)))$$

### Program:-

```
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
public class DesedeCrypter {
```



```
private static final String CRYPT_ALGORITHM = "DESede";

private static final String PADDING = "DESede/CBC/NoPadding";

private static final String CHAR_ENCODING = "UTF-8";

private static final byte[] MY_KEY = "5oquil2oo2vb63e8ionujny6".getBytes();//24-byte

private static final byte[] MY_IV = "3oco1v52".getBytes();//8-byte

public static void main(String[] args) {

    String srcText = "03215672";

    final DesedeCrypter crypter = new DesedeCrypter();

    String encryptedText = crypter.encrypt(srcText);

    System.out.println("sourceText=" + srcText + " -> encryptedText=" + encryptedText + "\n");
    System.out.println("encrypted-text=" + encryptedText + " -> decrypted-text(source text)="
        + crypter.decrypt(encryptedText));

}

public String encrypt(String text) {

    if (text == null) {

        return null;

    }

    String retVal = null;

    try {

        final SecretKeySpec secretKeySpec = new SecretKeySpec(MY_KEY, CRYPT_ALGORITHM);
        final IvParameterSpec iv = new IvParameterSpec(MY_IV);

        final Cipher cipher = Cipher.getInstance(PADDING);

        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, iv);
```

```
        final byte[] encrypted = cipher.doFinal(text.getBytes(CHAR_ENCODING));

        retVal = new String(encodeHex(encrypted));
    }

    catch (Exception e) {

        e.printStackTrace();
    }

    return retVal;
}

public String decrypt(String text) {

    if (text == null) {

        return null;
    }

    String retVal = null;

    try {

        final SecretKeySpec secretKeySpec = new SecretKeySpec(MY_KEY, CRYPT_ALGORITHM);

        final IvParameterSpec iv = new IvParameterSpec(MY_IV);

        final Cipher cipher = Cipher.getInstance(PADDING);
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, iv);

        final byte[] decrypted = cipher.doFinal(decodeHex(text.toCharArray()));

        retVal = new String(decrypted, CHAR_ENCODING);
    }

    catch (Exception e) {

        e.printStackTrace();
    }
}
```

```
        return retVal;
    }

    private byte[] decodeHex(char[] data) throws Exception {
int len = data.length;

    if ((len & 0x01) != 0) {
        throw new Exception("Odd number of characters.");
    }

    byte[] out = new byte[len >> 1];

    for (int i = 0, j = 0; j < len; i++) {
        int f = toDigit(data[j], j) << 4;

        j++;

        f = f | toDigit(data[j], j);

        j++;

        out[i] = (byte) (f & 0xFF);
    }

    return out;
}

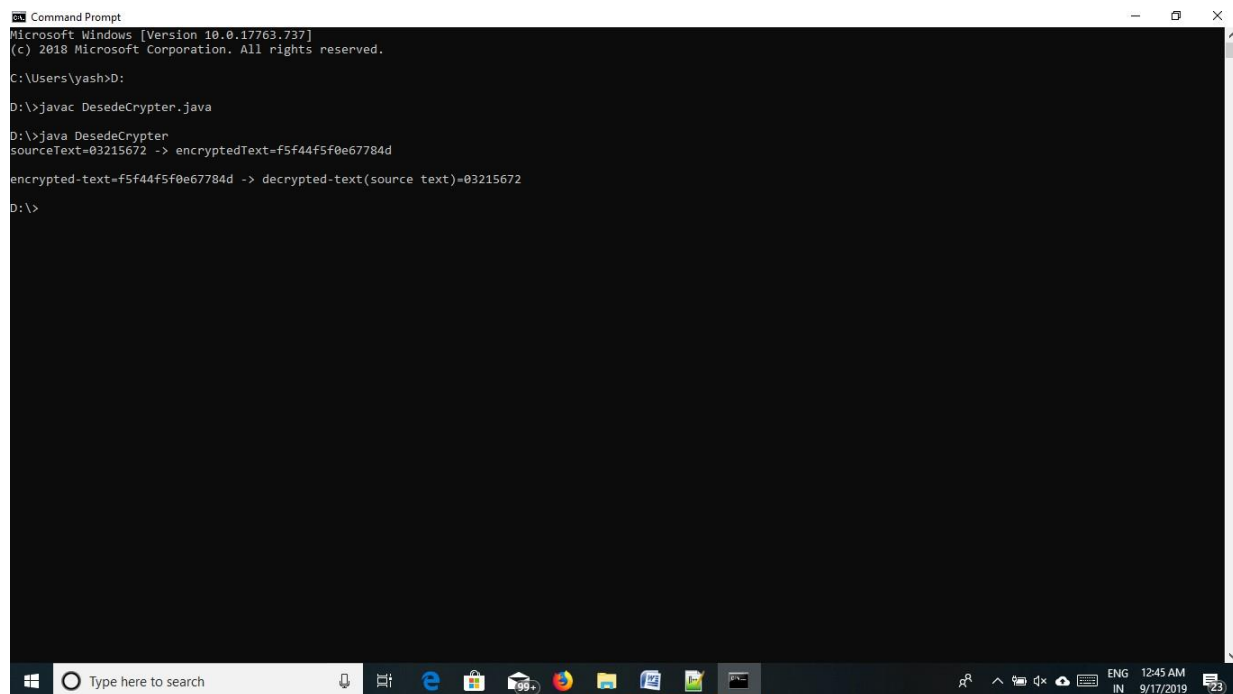
private int toDigit(char ch, int index) throws Exception {
    int digit = Character.digit(ch, 16);

    if (digit == -1) {
        throw new Exception("Illegal hexadecimal character " + ch + " at index " + index);
    }

    return digit;
}
```

```
private char[] encodeHex(byte[] data) {  
  
    final char[] DIGITS = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };  
  
    int l = data.length;  
  
    char[] out = new char[l << 1];  
  
    for (int i = 0, j = 0; i < l; i++) {  
  
        out[j++] = DIGITS[(0xF0 & data[i]) >>> 4];  
  
        out[j++] = DIGITS[0x0F & data[i]];  
  
    }  
  
    return out;  
  
}  
  
}
```

Op:-



```
Command Prompt  
Microsoft Windows [Version 10.0.17763.737]  
(c) 2018 Microsoft Corporation. All rights reserved.  
  
C:\Users\yash>D:  
  
D:\>javac DesedeCrypter.java  
  
D:\>java DesedeCrypter  
sourceText=03215672 -> encryptedText=f5f44f5f0e67784d  
  
encrypted-text=f5f44f5f0e67784d -> decrypted-text(source text)=03215672  
  
D:\>
```