Name :- Rohan Babulnath Kamble

Class :- T. Y. B. Sc. CS

PRN NO :- 2020420004

Subject :- Information And Network Security (USCSP504)

INDEX

Practical No:- 1

Aim:- Write programs to implement the following Substitution Cipher Techniques:-

a) Caesar Cipher b) Monoalphabetic Cipher

Theory:-

a) Caesar Cipher:-

Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.

For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

Program: -

```
import java.util.Scanner;
public class Caesar
{
    public static final String ALPHABET=  "abcdefghijklmnopqrstuvwxyz";
    public static String encryptData(String inputStr, int shiftKey)
   {
      inputStr = inputStr.toLowerCase();
      String encryptStr = "";
      for (int i = 0; i < inputStr.length(); i++)
      {
            int pos = ALPHABET.indexOf(inputStr.charAt(i));
            int encryptPos = (shiftKey + pos) % 26;
            char encryptChar = ALPHABET.charAt(encryptPos);
```

```
            encryptStr += encryptChar;
        }
         return encryptStr;
    }
public static String decryptData(String inputStr, int shiftKey)
{
           inputStr = inputStr.toLowerCase();
           String decryptStr = " ";
           for (int i = 0; i < inputStr.length(); i++)
          {
             int pos = ALPHABET.indexOf(inputStr.charAt(i));
            int decryptPos = (pos - shiftKey) % 26;
                if (decryptPos < 0)
               {
                    decryptPos = ALPHABET.length() + decryptPos;
                }
             char decryptChar = ALPHABET.charAt(decryptPos);
             decryptStr += decryptChar;
          }
          return decryptStr;
 }
    public static void main(String[] args)
   {
     Scanner sc = new Scanner(System.in);
     System.out.println("Enter a string for encryption using Caesar Cipher:");
     String inputStr = sc.nextLine();
    System.out.println("Enter the value by which each character in the plaintext
message gets shifted: ");
```

```
    int shiftKey = Integer.valueOf(sc.nextLine());
    System.out.println("Encrypted Data ===> "+encryptData(inputStr, shiftKey));
    System.out.println("Decrypted Data ===> "+decryptData(encryptData(inputStr,
shiftKey), shiftKey));
    sc.close();
   }
}
```

Output:-

```
Enter a string for encryption using Caeser Cipher:
sarita
Enter the value by which each character in the plaintext message gets shifted: 4
Encrypted Data ===> wevmxe
Decrypted Data===> sarita
```

b) Monoalphabetic Cipher:-

Theory:-

Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if ‘A’ is encrypted as ‘D’, for any number of occurrence in that plaintext, ‘A’ will always get encrypted to ‘D’.

Program:-

```
import java.io.*;
class monoalphabetic
{
    public static char normalChar[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
    'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };
    public static char codedChar[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P',
    'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C', 'V', 'B', 'N', 'M' };
    public static String stringEncryption(String s)
   {
       String encryptedString = "";
      for (int i = 0; i < s.length(); i++)
     {
          for (int j = 0; j < 26; j++)
         {
           if (s.charAt(i) == normalChar[j])
           {
              encryptedString += codedChar[j];
              break;
           }
           if (s.charAt(i) < 'a' || s.charAt(i) > 'z')
```

```
            {
             encryptedString += s.charAt(i);
             break;
            }
       }
  }
    return encryptedString;
}
public static String stringDecryption(String s)
{
      String decryptedString = " ";
      for (int i = 0; i < s.length(); i++)
      {
            for (int j = 0; j < 26; j++)
            {
                  if (s.charAt(i) == codedChar[j])
                  {
                        decryptedString += normalChar[j];
                        break;
                  }
                  if (s.charAt(i) < 'A' || s.charAt(i) > 'Z')
                 {
                      decryptedString += s.charAt(i);
                      break;
                  }
            }
      }
```

```
        return decryptedString;
    }
     public static void main(String args[])
    {
          String str = "Welcome to geeksforgeeks";
          System.out.println("Plain text: " + str);
          String encryptedString = stringEncryption(str.toLowerCase());
          System.out.println("Encrypted message: "+ encryptedString);
          System.out.println("Decrypted message: "
          +stringDecryption(encryptedString));
    }
}
```

Output:-

```
Plain text: Welcome to geeksforgeeks

Encrypted message: VTSEGDT ZG UTTALYGKUTTAL

Decrypted message: welcome to geeksforgeeks
```

Practical No:- 2

Aim:- Write programs to implement the following Transposition Cipher

Techniques:-

a) Vernam Cipher b) Playfair Cipher

a) Vernam Cipher:-

Theory:-

Vernam Cipher is a method of encrypting alphabetic text. It is one of the Substitution techniques for converting plain text into cipher text. In this mechanism we assign a number to each character of the Plain-Text, like (a = 0, b = 1, c = 2, … z = 25).

In the Vernam cipher algorithm, we take a key to encrypt the plain text whose length should be equal to the length of the plain text.

Program:-

```
import java.io.*;
public class VernamCipher
{
    public static int getCharValue(char x)
    {
        int y=(int)'a';
        return ((int)x-y);
    }
    public static char getNumberValue(int x)
    {
        int z=x+(int)'a';
        return ((char)z);
    }
    public static void main(String args[])throws Exception
```

```
{
  BufferedReader br=new BufferedReader(new
  InputStreamReader(System.in));
  System.out.println("Enter your plain text:");
  String accept=br.readLine();
  System.out.println("\nEnter your one time pad text:");
  String pad=br.readLine();
  int aval[]=new int[accept.length()];
  int pval[]=new int[pad.length()];
  int initval[]=new int[pad.length()];
  if(pad.length()!=accept.length())
 {
     System.out.println("Invalid one time pad. Application terminates.");
     return;
 }
 for(int i=0;i<accept.length();i++)
{
     int k=getCharValue(accept.charAt(i));
     aval[i]=k;
 }
 for(int i=0;i<pad.length();i++)
 {
     int k=getCharValue(pad.charAt(i));
     pval[i]=k;
 }
 for(int i=0;i<pad.length();i++)
 {
```

```
            initval[i]=aval[i]+pval[i];
            if(initval[i]>25)
            initval[i]-=26;
        }
         System.out.println("\nCipher text is : ");
         String cipher="";
         for(int i=0;i<pad.length();i++)
        {
             cipher+=getNumberValue(initval[i]);
         }
         System.out.print(cipher);
    }
}
```

Output:-

```
Enter your plain text:
 tycs
Enter your one time pad text:
 abcd
Cipher text is: tzev
```

b) Playfair Cipher-

Theory:-

Playfair cipher or Playfair Square is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a digraph (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then J is replaced by I. It means treat I and J as the same letter, accordingly. Since Playfair cipher encrypts the message digraph by digraph. Therefore, the Playfair cipher is an example of a digraph substitution cipher.

Program:-

```
import java.util.*;
class Basic
{
    String allchar="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    boolean indexofchar(char c)
   {
      for (int i = 0; i < allchar.length(); i++) {
      if(allchar.charAt(i)==c)
     {
       return true;
     }

   }
    return false;
}
```

```
}
   public class playfair
  {
        Basic b=new Basic();
        char keyMatrix[][]=new char[5][5];
        boolean repeat(char c)
     {
          if(!b.indexofchar(c))
         {
            return true;
          }
            for (int i = 0; i <keyMatrix.length; i++)
           {
                for (int j = 0; j < keyMatrix[i].length; j++)
               {

                     if(keyMatrix[i][j]==c || c=='J')
                    {
                      return true;
                    }
               }
      }
       return false;
}
     void insertKey(String key)
    {
        key=key.toUpperCase();
```

```
  key=key.replaceAll("J", "I");
  key=key.replaceAll("", "");
  int a=0,b=0;

  for (int k = 0; k < key.length(); k++)
{
     if(!repeat(key.charAt(k)))
    {
        keyMatrix[a][b++]=key.charAt(k);
        if(b>4)
       {
          b=0;
          a++;
       }
    }
}

 char p='A';
 while (a<5)
{
   while (b<5)
  {
      if (!repeat(p))
     {
        keyMatrix[a][b++]=p;
      }
       p++;
```

```
        }
         b=0;
         a++;
      }

    System.err.print("--------------------key matrix");
    for (int i = 0; i < 5; i++)
   {
         System.out.println("");
         for (int j = 0; j < 5; j++)
        {
              System.out.print("\t"+keyMatrix[i][j]);
        }

    }
    System.out.println("\n------------");
}
  int rowpos(char c)
{
        for (int i = 0; i < keyMatrix.length; i++)
       {
            for (int j = 0; j < keyMatrix[i].length; j++)
           {
              if(keyMatrix[i][j]==c)
             {
                  return i;
             }
```

```
        }
    }
     return -1;
}
    int columnpos(char c)
    {
      for (int i = 0; i < keyMatrix.length; i++)
     {
         for (int j = 0; j < keyMatrix[i].length; j++)
        {
            if(keyMatrix[i][j]==c)
            {
                  return j;
            }
        }

     }
      return -1;
}
    String encryptchar(String plain)
    {
         plain=plain.toUpperCase();
         char a=plain.charAt(0),b=plain.charAt(1);
         String ciphertext="";
         int r1,c1,r2,c2;
         r1=rowpos(a);
         c1=columnpos(a);
```

```
      r2=rowpos(b);
      c2=columnpos(b);

  if(c1==c2)
 {
     ++r1;
     ++r2;
      if(r1>4)
     {
         r1=0;
     }
   if(r2>4)
   {
       r2=0;
    }
   ciphertext+=keyMatrix[r1][c2];
   ciphertext+=keyMatrix[r2][c1];
}
   else if(r1==r2)
  {
       ++c1;
       ++c2;
        if(c1>4)
        c1=0;
        if(c2>4)
        c2=0;
        ciphertext+=keyMatrix[r1][c2];
```

```
      ciphertext+=keyMatrix[r2][c1];
}
  else
 {
    ciphertext+=keyMatrix[r1][c2];
    ciphertext+=keyMatrix[r2][c1];
 }
   return ciphertext;
}
  String Encrypt(String plaintext,String key)
 {
    insertKey(key);
    String ciphertext="";
    plaintext=plaintext.replaceAll("j", "i");
    plaintext=plaintext.replaceAll("", "");
    plaintext=plaintext.toUpperCase();
    int len=plaintext.length();

    if(len/2!=0)
   {
         plaintext+="X";
         ++len;
    }
     for (int i = 0; i < len-1; i=i+2)
    {
        ciphertext+=encryptchar(plaintext.substring(i,i+2));
        ciphertext+="";
```

```
    }
      return ciphertext;
  }
 String decryptchar(String cipher)
{
    cipher=cipher.toUpperCase();
    char a=cipher.charAt(0),b=cipher.charAt(1);
    String plainchar="";
    int r1,c1,r2,c2;
    r1=rowpos(a);
    c1=columnpos(a);
    r2=rowpos(b);
    c2=columnpos(b);

    if(c1==c2)
  {
       --r1;
       --r2;
       if(r1<0)
         r1=4;
       if(r2<0)
         r2=4;
       plainchar+=keyMatrix[r1][c2];
       plainchar+=keyMatrix[r2][c1];
  }
   else if(r1==r2)
  {
```

```
    --c1;
    --c2;
    if(c1<0)
      c1=4;
    if(c2<0)
      c2=4;
     plainchar+=keyMatrix[r1][c1];
     plainchar+=keyMatrix[r2][c2];
  }

      else
    {
        plainchar+=keyMatrix[r1][c2];
        plainchar+=keyMatrix[r2][c1];
    }
    return plainchar;
  }

String Decrypt(String ciphertext,String key)
{
   String plaintext="";
   ciphertext=ciphertext.replaceAll("j", "i");
   ciphertext=ciphertext.replaceAll("", "");
   ciphertext=ciphertext.toUpperCase();
   int len=ciphertext.length();
   for (int i = 0; i < len-1; i=i+2)
   {
```

```
            plaintext+=decryptchar(ciphertext.substring(i,i+2));
            plaintext+="";
        }
         return plaintext;
    }

     public static void main(String[] args)
     {
          playfair p=new playfair();
          Scanner sc=new Scanner(System.in);
          String key,ciphertext,plaintext;

         System.out.println("Enter plaintext:");
         plaintext=sc.nextLine();
         System.out.println("Enter key:");
         key=sc.nextLine();
         ciphertext=p.Encrypt(plaintext, key);
         System.out.println("Encrypted text");
         System.out.println("--\n"+ciphertext);
         System.out.println("----");
         String encryptedText=p.Decrypt(ciphertext, key);
         System.out.println("Decrypted text");
         System.out.println("------\n"+encryptedText);
         System.out.println("-------------\n");
    }
}
```

Output:-

Enter plaintext:

TYCSCLASS

Enter key:

NETWORK

N A B C D

E F G H I

K L M O P

Q R S T U

V W X Y Z

Encrypted text:

--

YCBTAOBRXB

Decrypted text:

TYCSCLASSX

----------key matrix

Practical No:-3

Aim:- Write programs to implement the following Transposition Cipher

Techniques:-

a) Rail Fence Cipher b) Simple Columnar Technique

a) Rail Fence Cipher:-

Theory:-

The rail fence cipher (sometimes called zigzag cipher) is a transposition cipher that jumbles up the order of the letters of a message using a basic algorithm.

The rail fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn.

For example, let's consider the **plaintext** "This is a secret message".

Plaintext T H I S I S A S E C R E T M E S S A G E

To encode this message we will first write over two lines (the "rails of the fence") as follows:

T		I		I		A		E		R		T		E		S		G	
	H		S		S		S		C		E		M		S		A		E

Rail Fence

Encoding

Note that all white spaces have been removed from the plain text.

The Cipher Text is then read off by writing the top row first, followed by the bottom row:

Ciphertext T I I A E R T E S G H S S S C E M S A E

Program:-

```
import java.io.*;
public class Railfence
{
    public static void main(String[] args)throws IOException
   {
        String pt,ct="";
        pt="TYCSBNB";
        String even="",odd="";
        for(int i=0;i<pt.length();i++)3
       {
            if(i%2==0)
           {
               even=even+pt.charAt(i);
            }
             else
             {
                odd=odd+pt.charAt(i);
              }
              ct=even+odd;
          }
          System.out.println("Plain Text:" +pt);
          System.out.println("Cipher Text:" +ct);
     }
}
```

Output:-

Plain Text: TYCSBNB

Cipher Text: TCBBYSN

b) Simple Columnar Technique:-

Theory:-

Columnar Transposition involves writing the plaintext out in rows , and then reading the ciphertext off in in columns.

We first pick a keyword for our encryption. We write the plaintext out in a grid where the number of columns is the number of letters in the keyword. We then title each column with the respective letter from the keyword. We take the letters in the keyword in alphabetical order, and read down the columns in this order. If a letter is repeated, we do the one that appears first, then the next and so on.

As an example, let's encrypt the message "The tomato is a plant in the nightshade family" using the keyword tomato. We get the grid given below.

T	**O**	**M**	**A**	**T**	**O**
5	**3**	**2**	**1**	**6**	**4**
T	H	E	T	O	M
A	T	O	I	S	A
P	L	A	N	T	I
N	T	H	E	N	I
G	H	T	S	H	A
D	E	F	A	M	I
L	Y	X	X	X	X

We have written the keyword above the grid of the plaintext, and also the numbers telling us which order to read the columns in. Notice that the first "O" is 3 and the second "O" is 4, and the same thing for the two "T"s.

Starting with the column headed by "A", our ciphertext begins "TINESAX" from this column. We now move to the column headed by "M", and so on through the letters of the keyword in alphabetical order to get the ciphertext "TINESAX / EOAHTFX / HTLTHEY / MAIIAIX / TAPNGDL / OSTNHMX" (where the / tells you where a new column starts). The final ciphertext is thus "TINES AXEOA HTFXH TLTHE YMAII AIXTA PNGDL OSTNH MX".

Program:-

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class simplecolumnar
{
    public static void main(String[] args) throws IOException
   {
        String plaintext ,ciphertext=" ",temp=" ";
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
        int i=0,j=0;
        System.out.println("Enter plaintext");
        plaintext=br.readLine();
        int plen=plaintext.length();
        for (i=0;i<plen;i++)
       {
            if(plaintext.charAt(i)!=' ')
            {
                temp+=plaintext.charAt(i);
            }
        }
        int len=temp.length();
        System.out.println("Enter no. of columns max."+len+"):");
```

```
 int cols=Integer.parseInt(br.readLine());
 if(cols<=len)
{
     int rows=1,k=0;
     if(len%cols==0)
    {
         rows=len/cols;
     }
     else
     {
         rows=(len/cols)+1;
     }
     char matrix[][]=new char[rows][cols];
     for(i=0;i<rows;i++)
    {
         for (j = 0; (j<cols)&&(k<len); j++)
        {
             matrix[i][j]=temp.charAt(k);
             k++;
         }
     }
       i--;
       for(;j<cols;j++)
      {
          matrix[i][j]='$';
      }
        for(i=0;i<rows;i++)
```

```
 {
    for(j=0;j<cols;j++)
   {
       System.out.print(matrix[i][j]+"");
   }
   System.out.println();
}
 int enCol[]=new int[cols];
 for(i=0;i<cols;i++)
{
   System.out.println("Enter column no."+(i+1)+":");
   enCol[i]=Integer.parseInt(br.readLine())-1;
}
  if(enCol.length==cols)
 {
     for(i=0;i<cols;i++)
    {
       for(j=0;j<rows;j++)
      {
          ciphertext+=matrix[j][enCol[i]];
      }
    }
 }
  temp=" ";
  for(i=0;i<ciphertext.length();i++)
 {
      if(ciphertext.charAt(i)!='$')
```

```
                {
                 temp+=ciphertext.charAt(i);
                }
              }
               ciphertext=temp;
               System.out.println("Ciphertext: "+ciphertext);
    }
     else
    {
       System.out.println("Incorrect no. of Columns");
     }
   }
}
```

Output:-

Enter your plain text:

tycs

Enter no. of columns max length is 4

tyc

s$$

Enter column no. 1:

2

Enter column no. 2:

1

Enter column no. 3:

3

Cipher text: ytsc

Practical No.:- 4

Aim:- Write program to encrypt and decrypt strings using

a) DES Algorithm b) AES Algorithm

a) DES Algorithm:-

Theory:-

DES is the archetypal block cipher—an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits. Like other block ciphers, DES by itself is not a secure means of encryption, but must instead be used in a mode of operation.

Decryption uses the same structure as encryption, but with the keys used in reverse order**.**

Program:-

```
import java.io.UnsupportedEncodingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
public class DesEncrypter
{
    Cipher ecipher;
    Cipher dcipher;
```

```
 DesEncrypter(SecretKey key)
{
     try
    {
         ecipher=Cipher.getInstance("DES");
         dcipher=Cipher.getInstance("DES");
         ecipher.init(Cipher.ENCRYPT_MODE,key);
         dcipher.init(Cipher.DECRYPT_MODE,key);
    }
     catch(javax.crypto.NoSuchPaddingException e){}
     catch(java.security.NoSuchAlgorithmException e){}
     catch(java.security.InvalidKeyException e){}
}
 public String encrypt(String str)
{
     try
     {
         byte[] utf8=str.getBytes("Utf8");
         byte[] enc=ecipher.doFinal(utf8);
         return new sun.misc.BASE64Encoder().encode(enc);
     }
     catch(javax.crypto.BadPaddingException e){}
     catch(IllegalBlockSizeException e){}
     catch(UnsupportedEncodingException e){}
     catch(java.io.IOException e){}
     return null;
 }
```

```
 public String decrypt(String str)
{
    try
   {
        byte[] dec=new sun.misc.BASE64Decoder().decodeBuffer(str);
        byte[] utf8=dcipher.doFinal(dec);
        return new String(utf8,"Utf8");
    }
    catch(javax.crypto.BadPaddingException e){}
    catch(IllegalBlockSizeException e){}
    catch(UnsupportedEncodingException e){}
    catch(java.io.IOException e){}
    return null;
 }
 public static void main(String[] args)
{
    System.out.println();
    System.out.println("------*---Encrypting String using DES---*------");
    System.out.println();
    try
   {
        SecretKey key=KeyGenerator.getInstance("DES").generateKey();
        DesEncrypter encrypter=new DesEncrypter(key);
        String s="Don't tell Anybody!!";
        String d="Hello";
        String encrypted=encrypter.encrypt(s);
        String decrypted=encrypter.decrypt(encrypted);
```

```
            System.out.println("Original String is:"+s);
            System.out.println("Encrypted String is:"+encrypted);
            System.out.println("Decrypted String is:"+decrypted);
        }
        catch(Exception e){}
    }
}
```

Output:-

```
-------*----Encrypting String using DES----*-------
Original String is :Don’t tell Anybody!!
Encrypted String is:RGJNJCy4EuOm+B8983/A2AbvDoXQmpjo
Decrypted String is :Don’t tell Anybody!!
```

b) AES Algorithm:-

Theory:-

AES is based on a design principle known as a substitution–permutation network, and is efficient in both software and hardware.

AES operates on a 4 × 4 column-major order array of bytes, termed the state. Most AES calculations are done in a particular finite field. For instance, if there are 16 bytes, b0 , b1 , b2……b16 , these bytes are represented as this two-dimensional array:

b0 b4 b8 b12

b1 b5 b9 b13

b2 b6 b10 b14

b3 b7 b11 b15

The key size used for an AES cipher specifies the number of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of rounds are as follows:

- 10 rounds for 128-bit keys.
- 12 rounds for 192-bit keys.
- 14 rounds for 256-bit keys.

Each round consists of several processing steps, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

Program:-

```
 import java.security.*;
import javax.crypto.*;
import java.io.*;
public class AES_StringEncrypter
{
    Cipher ecipher;
    Cipher dcipher;
    AES_StringEncrypter(SecretKey key)
```

```
{
    try
   {
        ecipher=Cipher.getInstance("AES");
        dcipher=Cipher.getInstance("AES");
        ecipher.init(Cipher.ENCRYPT_MODE,key);
        dcipher.init(Cipher.DECRYPT_MODE,key);
    }
    catch(Exception e)
   {
    }
 }
 public String encrypt(String str)
{
    try
   {
        byte[] utf8=str.getBytes("UTF-8");
        byte[] enc=ecipher.doFinal(utf8);
        return new sun.misc.BASE64Encoder().encode(enc);
    }
    catch(Exception e)
   {
    }
    return null;
 }
 public String decrypt(String str)
{
    try
   {
        byte[] dec=new sun.misc.BASE64Decoder().decodeBuffer(str);
        byte[] utf8=dcipher.doFinal(dec);
```

```
            return new String(utf8,"UTF8");
        }
        catch(Exception e)
       {
        }
         return null;
    }
     public static void main(String[] args)
{
        SecretKey key=null;
        try
       {
            KeyGenerator keyGen=KeyGenerator.getInstance("AES");
            key=keyGen.generateKey();
        }
         catch(Exception e)
        {
            e.printStackTrace();
        }
         AES_StringEncrypter dese=new AES_StringEncrypter(key);
         String o="abc";
         String en=dese.encrypt(o);
         String de=dese.decrypt(en);
         System.out.println("Original Text:"+o);
         System.out.println("Encrypted Text:"+en);
         System.out.println("Decrypted Text:"+de);
    }
}
```

Output:-

Original Text: abc

Encrypted Text: VaH1PmeW0XAvZt8i7CY1fQ==

Decrypted Text: abc

Practical No.:- 5

Aim:- Write a program to implement RSA algorithm to perform encryption / decryption of a given string.

RSA Algorithm:-

Theory:-

The RSA cryptosystem is the most widely-used public key cryptography algorithm in the world. It can be used to encrypt a message without the need to exchange a secret key separately.

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Program:-

```
import java.math.BigInteger;
import java.security.SecureRandom;
public class RSA
{
    private BigInteger n,d,e;
    private int bitlen=1024;
    public RSA(BigInteger newn,BigInteger newe)
   {
        n=newn;
        e=newn;
    }
     public RSA(int bits)
    {
        bitlen=bits;
        SecureRandom r=new SecureRandom();
```

```
    BigInteger p=new BigInteger(bitlen/2,100,r);
    BigInteger q=new BigInteger(bitlen/2,100,r);
    n=p.multiply(q);
    BigInteger
    m=(p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
    e=new BigInteger("3");
    while(m.gcd(e).intValue()>1)
   {
        e=e.add(new BigInteger("2"));
    }
    d=e.modInverse(m);
 }
 public synchronized String encrypt(String message)
{
    return (new BigInteger(message.getBytes())).modPow(e,n).toString();
 }
 public synchronized BigInteger encrypt(BigInteger message)
{
    return message.modPow(e, n);
 }
 public synchronized String decrypt(String message)
{
    return new String((new
  BigInteger(message)).modPow(d,n).toByteArray());
 }
 public synchronized BigInteger decrypt(BigInteger message)
 {
```

```
    return message.modPow(d, n);
 }
 public synchronized void generateKeys()
{
   SecureRandom r=new SecureRandom();
   BigInteger p= new BigInteger(bitlen/2,100,r);
   BigInteger q=new BigInteger(bitlen/2,100,r);
   n=p.multiply(q);
   BigInteger m =
    (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
     e=new BigInteger("3");
     while(m.gcd(e).intValue()>1)
    {
        e=e.add(new BigInteger("2"));
     }
     d=e.modInverse(m);
 }
 public synchronized BigInteger getN()
{
    return n;
 }
 public synchronized BigInteger getE()
{
    return e;
 }
 public static void main(String[] args)
{
```

```
        RSA rsa=new RSA(1024);
        String text1="Yellow and Black";
        System.out.println("Plain Text:"+text1);
        BigInteger plaintext=new BigInteger(text1.getBytes());
        BigInteger ciphertext=rsa.encrypt(plaintext);
        System.out.println("Cipher Text:"+ciphertext);
        plaintext=rsa.decrypt(ciphertext);
        String text2=new String(plaintext.toByteArray());
        System.out.println("Plaintext:"+text2);
    }
}
```

Output:-

Plain Text: Yellow and Black

Cipher Text:

23691488215321139460749997063202859980905398317494924806955921765907585336003472280091721878076037166070001206261426439834777158661530688970408124653219393512420829623699675204193197656418299

Plaintext: Yellow and Black

Practical No:- 6

Aim:- Write a program to implement the Diffie-Hellman Key Agreement algorithm to generate symmetric keys.

Theory:

Diffie–Hellman key exchange (DH) is a method of securely exchanging cryptographic keys over a public channel.

Step 1: A and B agree on two large Prime Numbers ‘N’ and ‘G’ respectively.

Step 2: A select another large random number ‘x’ and calculate A such that

$A= (G^x) \bmod N$

Step 3: Sender send the number ‘A’ to receiver.

Step 4: B select another large random number ‘y’ and calculate B such that

$B= (G^y) \bmod N$

Step 5: Receiver send the number ‘B’ to sender.

Step 6: Sender now compute Secret Key K1 such that $K1= (B^x) \bmod N$

Step 7: Receiver now compute Secret Key K2 such that $K2= (A^y) \bmod N$

Program:-

```
import java.math.BigInteger;
import java.util.*;
public class DiffieHellman
{
   public static void main(String[] args)
  {
        Scanner sc=new Scanner(System.in);
        BigInteger n,g,x,y,k1,k2,A,B;
        System.out.println("Enter two Prime Numbers: ");
```

```
        n=new BigInteger(sc.next());
        g=new BigInteger(sc.next());
        System.out.println("Person A:Enter your Secret Number");
        x=new BigInteger(sc.next());
        A=g.modPow(x,n);
        System.out.println("Person B:Enter your Secret Number");
        y=new BigInteger(sc.next());
        B=g.modPow(y,n);
        k1=B.modPow(x,n);
        k2=A.modPow(y,n);
        System.out.println("A's Secret Key:" +k1);
        System.out.println("B's Secret Key:" +k2);
    }
}
```

Output:-

Enter two Prime Numbers:-

13

17

Person A:Enter your Secret Number-

5

Person B:Enter your Secret Number-

8

A’s Secret Key:-9

B’s Secret Key:-9

Practical No:- 7

Aim:- Write a program to implement the MD5 algorithm compute the message digest.

Theory:-

The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message.

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit words); the message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits fewer than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 264.

Program:-

```
import java.util.Scanner;
import javax.xml.bind.DatatypeConverter;
import java.security.MessageDigest;
public class MD5
{
    public static void main(String[] args)
    {
        Scanner sn=new Scanner(System.in);
        System.out.print("Please Enter Data for which MD5 is required:");
        String data=sn.nextLine();
```

```
        MD5 md=new MD5();
        String hash=md.getMD5Hash(data);
        System.out.println("The MD5 (hexadecimalencoded)hash is:"+hash);
    }
    private String getMD5Hash(String data){
        String result=null;
        try
       {
          MessageDigest digest=MessageDigest.getInstance("MD5");
          byte[] hash=digest.digest(data.getBytes("UTF-8"));
          return bytesToHex(hash);
        }
         catch(Exception ex)
        {
            ex.printStackTrace();
        }
          return result;
    }
    private String bytesToHex(byte[] hash)
   {
        return DatatypeConverter.printHexBinary(hash);
    }
}
```

Output:-

Please Enter Data for which MD5 is required:

Welcome to TYCS

The MD5(hexadecimalencoded)hash is:-
C84A98451A48AA000A03E86BA49DAFE7

Practical No:- 8

Aim:- Write a program to calculate HMAC-SHA1 Signature.

Theory:-

In cryptography, an HMAC (sometimes expanded as either keyed-hash message authentication code or hash-based message authentication code) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.

SHA-1 is one of the hash function which produces a hash digest of 160 bits (20 bytes).

Documents may refer to SHA-1 as just "SHA", even though this may conflict with the other Standard Hash Algorithms such as SHA-0, SHA-2 and SHA-3.

HMAC does not encrypt the message. Instead, the message (encrypted or not) must be sent alongside the HMAC hash. Parties with the secret key will hash the message again themselves, and if it is authentic, the received and computed hashes will match.

Program:-

```
import java.security.SignatureException;

import java.util.*;

import java.security.*;

import javax.crypto.*;

import java.security.InvalidKeyException;

import javax.crypto.spec.*;

import javax.crypto.spec.SecretKeySpec;

public class Msgobj

{
```

```
 private static final String HMAC_SHA1_ALGORITHM="hMACsha1";
   private static String toHexString(byte[] bytes)
{
     Formatter formatter=new Formatter();
     for(byte b:bytes)
      {
          formatter.format("%02x", b);
       }
       return formatter.toString();
}
 public static  String calculateRFC2104HMAC(String data,String key)throws
SignatureException,
NoSuchAlgorithmException,InvalidKeyException
{
  SecretKeySpec signingKey=new
       SecretKeySpec(key.getBytes(),HMAC_SHA1_ALGORITHM);
       Mac mac=Mac.getInstance(HMAC_SHA1_ALGORITHM);
       mac.init(signingKey);
       return toHexString(mac.doFinal(data.getBytes()));
  }
   public static void main(String[] args)throws Exception
  {
     String hmac=calculateRFC2104HMAC("data","key") ;
     System.out.println(hmac);
```

```
    }
}
```

Output:-

```
104152c5bfdca07bc633eebd46199f0255c9f49d
```