SHIFTCIPHER

```
import java.util.*;
class Main {
  static String encrypt(String plaintext, int key)
  {
    String ciphertext="";
    for (int i=0;i<plaintext.length();i++)
    {
      char c = plaintext.charAt(i);
      int ascii = (int)c;
      if (c==' ')
        ciphertext+=c;
      else if (ascii>=65 && ascii<=90)
      {
        int n=(((ascii-65)+key)%26)+65;
        char c1 = (char)n;
        ciphertext+=c1;
      }
      else if (ascii>=97 && ascii<=122)
      {
        int n=(((ascii-97)+key)%26)+97;
        char c1 = (char)n;
        ciphertext+=c1;
      }
    }
    return ciphertext;
  }
  static String decrypt(String plaintext, int key)
  {
    String ciphertext="";
    for (int i=0;i<plaintext.length();i++)
    {
      char c = plaintext.charAt(i);
      int ascii = (int)c;
      if (c==' ')
        ciphertext+=c;
      else if (ascii>=65 && ascii<=90)
      {
        int n=(((ascii-65)-key)%26);
        while (n<0)
          n+=26;
        n+=65;
        char c1 = (char)n;
        ciphertext+=c1;
      }
      else if (ascii>=97 && ascii<=122)
```

```
    {
      int n=(((ascii-97)-key)%26);
      while (n<0)
        n+=26;
      n+=97;
      char c1 = (char)n;
      ciphertext+=c1;
    }
  }
  return ciphertext;
}
public static void main(String[] args) {
  String plaintext,ciphertext;
  System.out.println("Shift Cipher");
  System.out.println("\nEncryption");
  System.out.println("Enter plaintext: ");
  Scanner s = new Scanner(System.in);
  plaintext = s.nextLine();
  System.out.println("Enter key (between 0 and 25)");
  int key = s.nextInt();
  s.nextLine();
  if (key<0 || key>25)
  {
    System.out.println("Invalid key.");
    System.exit(0);
  }
  for (int i=0;i<plaintext.length();i++)
  {
    char c = plaintext.charAt(i);
    int ascii = (int)c;
    if (c!=' ')
    if (!((ascii<=90 && ascii>=65) || ((ascii)<=122) && ascii>=97))
    {
      System.out.println("Invalid plaintext");
      System.exit(0);
    }
  }
  ciphertext = encrypt(plaintext,key);
  System.out.println("Ciphertext: "+ciphertext);
  System.out.println("\nDecryption\nEnter ciphertext: ");
  ciphertext = s.nextLine();
  plaintext=decrypt(ciphertext,key);
  System.out.println("Plaintext: "+plaintext);
  System.out.println("Cryptanalysis: ");
  ArrayList<String> dict = new ArrayList<String>();
  ArrayList<Integer> posskeys = new ArrayList<Integer>();
  ArrayList<String> possplains = new ArrayList<String>();
  dict.add("dog");
```

```
    dict.add("cat");
    dict.add("zebra");
    System.out.println("\n\nKEY\t\tPLAINTEXT");
    for (int k=0;k<26;k++)
    {
      plaintext=decrypt(ciphertext,k);
      System.out.println(k+"\t\t"+plaintext);
      String[] words = plaintext.split("\\s");
      for (String word: words)
      {
        if (dict.contains(word))
        {
          if (!posskeys.contains(k))
          {
            posskeys.add(k);
            possplains.add(plaintext);
          }

        }
      }
    }
    System.out.println("Possible keys and plaintext found:");
    System.out.println("\n\nKEY\t\tPLAINTEXT");
    for (int i=0;i<posskeys.size();i++)
    {
      System.out.println(posskeys.get(i)+"\t\t"+possplains.get(i));
    }

  }
}
```

PLAYFAIRCIPHER

```
import java.util.*;

class Main {
  static char[][] keymatrix;

  static boolean hasOnlyAlphabets(String text) {
    boolean flag = true;
    for (int i = 0; i < text.length(); i++) {
      char c = text.charAt(i);
      if (!(c >= 97 && c <= 122))
        flag = false;
    }
    return flag;
  }

  static void genKeyMatrix(String keyword) {
    keymatrix = new char[5][5];
    boolean[] alphabetlist = new boolean[26];
    int keyindex = 0;
    int alphindex = 0;
    int fl = 0;
    int i = 0, j = 0;
    while (keyindex < keyword.length()) {
      keymatrix[i][j++] = keyword.charAt(keyindex);
      int n = (int) keyword.charAt(keyindex);
      n -= 97;
      alphabetlist[n] = true;
      keyindex++;
      if (j >= 5) {
        i++;
        j = 0;
      }
    }
    for (int in = 0; in < 26; in++) {
      if (in != 9) {
        if (alphabetlist[in] == false) {
          char c = (char) (in + 97);
          keymatrix[i][j++] = c;
          if (j >= 5) {
            i++;
            j = 0;
          }
        }
      }
```

```
  }
  System.out.println("\nKEY MATRIX: ");
  for (i = 0; i < 5; i++) {
    for (j = 0; j < 5; j++) {
      System.out.print(keymatrix[i][j] + "  ");
    }
    System.out.println();
  }
}

static String encrypt(String plaintext) {
  String ciphertext = "";
  int i1 = 0, i2 = 0, j1 = 0, j2 = 0;
  for (int i = 0; i < plaintext.length(); i += 2) {
    char c1 = plaintext.charAt(i);
    if (i + 1 < plaintext.length()) {
      char c2 = plaintext.charAt(i + 1);
      for (int j = 0; j < 5; j++) {
        for (int k = 0; k < 5; k++) {
          if (keymatrix[j][k] == c1) {
            i1 = j;
            j1 = k;
          } else if (keymatrix[j][k] == c2) {
            i2 = j;
            j2 = k;
          }
        }
      }
      if (i1 == i2) {
        j1 = (j1 + 1) % 5;
        j2 = (j2 + 1) % 5;
        char c3 = keymatrix[i1][j1];
        char c4 = keymatrix[i2][j2];
        ciphertext += c3;
        ciphertext += c4;
      } else if (j1 == j2) {
        i1 = (i1 + 1) % 5;
        i2 = (i2 + 1) % 5;
        char c3 = keymatrix[i1][j1];
        char c4 = keymatrix[i2][j2];
        ciphertext += c3;
        ciphertext += c4;
      } else {
        char c3 = keymatrix[i1][j2];
        char c4 = keymatrix[i2][j1];
        ciphertext += c3;
        ciphertext += c4;
```

```
      }
    }

  }
  return ciphertext;
}

static String decrypt(String plaintext) {
  String ciphertext = "";
  int i1 = 0, i2 = 0, j1 = 0, j2 = 0;
  for (int i = 0; i < plaintext.length(); i += 2) {
    char c1 = plaintext.charAt(i);
    if (i + 1 < plaintext.length()) {
      char c2 = plaintext.charAt(i + 1);
      for (int j = 0; j < 5; j++) {
        for (int k = 0; k < 5; k++) {
          if (keymatrix[j][k] == c1) {
            i1 = j;
            j1 = k;
          } else if (keymatrix[j][k] == c2) {
            i2 = j;
            j2 = k;
          }
        }
      }
      if (i1 == i2) {
        j1 = (j1 - 1) % 5;
        j2 = (j2 - 1) % 5;
        if (j1 < 0)
          j1 += 5;
        if (j2 < 0)
          j2 += 5;
        char c3 = keymatrix[i1][j1];
        char c4 = keymatrix[i2][j2];
        ciphertext += c3;
        ciphertext += c4;
      } else if (j1 == j2) {
        i1 = (i1 - 1) % 5;
        i2 = (i2 - 1) % 5;
        if (i1 < 0)
          i1 += 5;
        if (i2 < 0)
          i2 += 5;
        char c3 = keymatrix[i1][j1];
        char c4 = keymatrix[i2][j2];
        ciphertext += c3;
        ciphertext += c4;
      } else {
```

```
        char c3 = keymatrix[i1][j2];
        char c4 = keymatrix[i2][j1];
        ciphertext += c3;
        ciphertext += c4;
      }
    }

  }
  return ciphertext;
}

public static void main(String[] args) {
  String plaintext, ciphertext, keyword;
  System.out.println("\nPLAYFAIR CIPHER");
  System.out.println("\nENCRYPTION\nEnter plaintext:");
  Scanner s = new Scanner(System.in);
  plaintext = s.nextLine();
  System.out.println("Enter keyword: ");
  keyword = s.nextLine();
  if (!(hasOnlyAlphabets(plaintext)) || !(hasOnlyAlphabets(keyword))) {
    System.out.println("Only alphabets are allowed.");
    System.exit(0);
  }
  ArrayList<Character> kl = new ArrayList<Character>();
  for (int i = 0; i < keyword.length(); i++) {
    kl.add(keyword.charAt(i));
  }
  for (int i = 0; i < kl.size(); i++) {
    char c = kl.get(i);
    if (c == 'j')
      kl.set(i, 'i');
  }
  for (int i = 0; i < kl.size(); i++) {
    for (int j = i + 1; j < kl.size(); j++) {
      if (kl.get(j) == kl.get(i)) {
        kl.remove(j--);
      }
    }
  }
  keyword = "";
  for (char c : kl)
    keyword += c;
  genKeyMatrix(keyword);
  ArrayList<Character> pl = new ArrayList<Character>();
  for (int i = 0; i < plaintext.length(); i++) {
    if (plaintext.charAt(i) == 'j')
      pl.add('i');
    else
```

```
        pl.add(plaintext.charAt(i));
    }
    for (int i = 0; i < pl.size(); i += 2) {
      char c1 = pl.get(i);
      if (i + 1 < pl.size()) {
        char c2 = pl.get(i + 1);
        if (c1 == c2) {
          pl.add(i + 1, 'x');
        }
      }

    }
    while (pl.size() % 2 != 0)
      pl.add('x');
    plaintext = "";
    for (char c : pl)
      plaintext += c;
    System.out.println("\nModified plaintext: " + plaintext);
    ciphertext = encrypt(plaintext);
    System.out.println("\nCiphertext: " + ciphertext);
    System.out.println("\nDECRYPTION\nEnter ciphertext: ");
    ciphertext = s.nextLine();
    plaintext = decrypt(ciphertext);
    System.out.println("\nPlaintext: " + plaintext);
  }
}
```

HILLCIPHER

```
import java.util.*;
class Main {
  static int[][] genKeyMatrix(String keyword)
  {
    int[][] keymatrix = new int[3][3];
    int ind=0;
    for (int i=0;i<3;i++)
    {
      for (int j=0;j<3;j++)
      {
        keymatrix[i][j]=((int)(keyword.charAt(ind++)))-97;
      }
    }
    System.out.println("\nKEY MATRIX");
    for (int i=0;i<3;i++)
    {
      for (int j=0;j<3;j++)
      {
        System.out.print(keymatrix[i][j]+"  ");
      }
      System.out.println();
    }
    return keymatrix;
  }
  static int[][] multiplyMatrix(int[][] a, int[][] b)
  {
    int[][] r = new int[3][1];
    int m=3,n=3,p=3,q=1,k,sum=0;
    for(int i=0;i<m;i++)
    {
      for (int j=0;j<q;j++)
      {
        for (k=0;k<p;k++)
        {
          sum+=a[i][k]*b[k][j];
        }
        r[i][j]=sum;
        sum=0;
      }
    }
    return r;
  }
  static String encrypt(String plaintext, int[][] keymat)
  {
```

```
  String ciphertext="";
  int[][] trigraphmat = new int[3][1];
  int[][] result = new int[3][1];
  for (int i=0;i<plaintext.length();i+=3)
  {
    String trigraph = plaintext.substring(i,i+3);
    trigraphmat[0][0] = ((int)(trigraph.charAt(0)))-97;
    trigraphmat[1][0] = ((int)(trigraph.charAt(1)))-97;
    trigraphmat[2][0] = ((int)(trigraph.charAt(2)))-97;
    result = multiplyMatrix(keymat,trigraphmat);
    for (int in=0;in<3;in++)
    {
      int n = (result[in][0]%26)+97;
      char c = (char)(n);
      ciphertext+=c;
    }
  }
  return ciphertext;
}
static String decrypt(String plaintext, int[][] keymat)
{
  String ciphertext="";
  int[][] trigraphmat = new int[3][1];
  int[][] result = new int[3][1];
  for (int i=0;i<plaintext.length();i+=3)
  {
    String trigraph = plaintext.substring(i,i+3);
    trigraphmat[0][0] = ((int)(trigraph.charAt(0)))-97;
    trigraphmat[1][0] = ((int)(trigraph.charAt(1)))-97;
    trigraphmat[2][0] = ((int)(trigraph.charAt(2)))-97;
    result = multiplyMatrix(keymat,trigraphmat);
    for (int in=0;in<3;in++)
    {
      int n = (result[in][0]%26)+97;
      char c = (char)(n);
      ciphertext+=c;
    }
  }
  return ciphertext;
}
static int[][] findInverseMatrix(int[][] a)
{
  int[][] im = new int[3][3];
  int det;
  int v1 = a[0][0]*((a[1][1]*a[2][2]) - a[1][2]*a[2][1]);
  int v2 = a[0][1]*(a[1][0]*a[2][2] - a[1][2]*a[2][0]);
  int v3 = a[0][2]*(a[1][0]*a[2][1] - a[1][1]*a[2][0]);
  det = v1-v2+v3;
```

```
  det%=26;
  if (det<0)
  det+=26;
  int inv=0;
  for (int i=1;i<25;i++)
  {
    if ((det*i)%26==1)
      inv=i;
  }
  if (inv!=0)
  {
    int[] v = new int[9];
    v[0] = (a[1][1]*a[2][2] - a[1][2]*a[2][1]);
    v[1] = -(a[1][0]*a[2][2] - a[1][2]*a[2][0]);
    v[2] = (a[1][0]*a[2][1] - a[1][1]*a[2][0]);
    v[3] = -(a[0][1]*a[2][2] - a[0][2]*a[2][1]);
    v[4] = a[0][0]*a[2][2] - a[0][2]*a[2][0];
    v[5] = -(a[0][0]*a[2][1] - a[0][1]*a[2][0]);
    v[6] = a[0][1]*a[1][2] - a[0][2]*a[1][1];
    v[7] = -(a[0][0]*a[1][2] - a[0][2]*a[1][0]);
    v[8] = a[0][0]*a[1][1] - a[0][1]*a[1][0];
    int index=0;
    for (int j=0;j<3;j++)
    {
      for (int i=0;i<3;i++)
      {
        int n = (inv*v[index++])%26;
        if (n<0)
          n+=26;
        im[i][j] = n;
      }
    }
    System.out.println("\nINVERSE KEY MATRIX");
    for (int i=0;i<3;i++)
    {
      for (int j=0;j<3;j++)
      {
        System.out.print(im[i][j]+"  ");
      }
      System.out.println();
    }
  }
  else
    System.out.println("Inverse does not exist!");
  return im;
}
public static void main(String[] args) {
  String plaintext,ciphertext,keyword;
```

```
    System.out.println("\nHILL CIPHER");
    System.out.println("\nEnter plaintext: ");
    Scanner s = new Scanner(System.in);
    plaintext = s.nextLine();
    System.out.println("Enter keyword: ");
    keyword = s.nextLine();
    while (plaintext.length()%3!=0)
      plaintext+='x';
    int ascii=0;
    while (keyword.length()!=9)
      keyword+=(char)((ascii++)+97);
    System.out.println("\nNew plaintext: "+plaintext);
    System.out.println("New Keyword: "+keyword);
    int[][] keymatrix = genKeyMatrix(keyword);
    ciphertext = encrypt(plaintext,keymatrix);
    System.out.println("Ciphertext: "+ciphertext);
    System.out.println("\nDECRYPTION\nEnter ciphertext: ");
    ciphertext = s.nextLine();
    int[][] invkeymatrix = findInverseMatrix(keymatrix);
    plaintext = decrypt(ciphertext,invkeymatrix);
    System.out.println("Plaintext: "+plaintext);
  }
}
```

VIGNERECIPHER

```
import java.util.*;
class Main {
  static String encrypt (String plaintext, String keyword)
  {
    String ciphertext="";
    for (int i=0;i<plaintext.length();i++)
    {
      int v1 = (int)(plaintext.charAt(i));
      int v2 = (int)(keyword.charAt(i));
      v1-=97;
      v2-=97;
      int n = (v1+v2)%26;
      char c = (char)(n+97);
      ciphertext+=c;
    }
    return ciphertext;
  }
  static String decrypt (String plaintext, String keyword)
  {
    String ciphertext="";
    for (int i=0;i<plaintext.length();i++)
    {
      int v1 = (int)(plaintext.charAt(i));
      int v2 = (int)(keyword.charAt(i));
      v1-=97;
      v2-=97;
      int n = (v1-v2)%26;
      if (n<0)
        n+=26;
      char c = (char)(n+97);
      ciphertext+=c;
    }
    return ciphertext;
  }
  public static void main(String[] args) {
    String plaintext,ciphertext,keyword;
    System.out.println("\nVIGNERE CIPHER\n\nENCRYPTION\nEnter plaintext: ");
    Scanner s = new Scanner(System.in);
    plaintext = s.nextLine();
    System.out.println("Enter keyword: ");
    keyword = s.nextLine();
    int pl = plaintext.length();
    int kl = keyword.length();
    while (kl<pl)
```

```
    {
      keyword+=keyword;
      kl = keyword.length();
    }
    keyword = keyword.substring(0,pl);
    ciphertext = encrypt(plaintext,keyword);
    System.out.println("Ciphertext: "+ciphertext);
    System.out.println("\nDECRYPTION\nEnter ciphertext: ");
    ciphertext = s.nextLine();
    plaintext = decrypt(ciphertext,keyword);
    System.out.println("Plaintext: "+plaintext);

  }
}
```

RAILFENCECIPHER

```
import java.util.*;
class Main {
  static String encrypt(String plaintext, int key)
  {
    String ciphertext="";
    int pl = plaintext.length();
    char[][] perm = new char[key][pl];
    for (int i=0;i<key;i++)
    {
      for (int j=0;j<pl;j++)
      {
        perm[i][j]='*';
      }
    }
    int i=0;
    int j=0;
    boolean up=false;
    boolean down=true;
    for (int index=0;index<pl;index++)
    {
      perm[i][j] = plaintext.charAt(index);
      if (down)
      {
        i++;
        j++;
        if (i>=key)
        {
          i--;
          j--;
          i--;
          j++;
          up=true;
          down=false;
        }
      }
      else if (up)
      {
        i--;
        j++;
        if (i<0)
        {
          i++;
          j--;
          i++;
```

```
        j++;
        up=false;
        down=true;
      }
    }
  }
  for (i=0;i<key;i++)
  {
    for (j=0;j<pl;j++)
    {
      if (perm[i][j]!='*')
        ciphertext+=perm[i][j];
    }
  }
  return ciphertext;
}
static String decrypt(String plaintext, int key)
{
  String ciphertext="";
  int ind=0;
  int pl = plaintext.length();
  char[][] perm = new char[key][pl];
  for (int i=0;i<key;i++)
  {
    for (int j=0;j<pl;j++)
    {
      perm[i][j]='*';
    }
  }
  int i=0;
  int j=0;
  boolean up=false;
  boolean down=true;
  for (int index=0;index<pl;index++)
  {
    perm[i][j] = '-';
    if (down)
    {
      i++;
      j++;
      if (i>=key)
      {
        i--;
        j--;
        i--;
        j++;
        up=true;
        down=false;
```

```
    }
  }
  else if (up)
  {
    i--;
    j++;
    if (i<0)
    {
      i++;
      j--;
      i++;
      j++;
      up=false;
      down=true;
    }
  }
}
for (i=0;i<key;i++)
{
  for (j=0;j<pl;j++)
  {
    if (perm[i][j]=='-')
      perm[i][j]=plaintext.charAt(ind++);
  }
}
i=0;
j=0;
up=false;
down=true;
for (int index=0;index<pl;index++)
{
  ciphertext+=perm[i][j];
  if (down)
  {
    i++;
    j++;
    if (i>=key)
    {
      i--;
      j--;
      i--;
      j++;
      up=true;
      down=false;
    }
  }
  else if (up)
  {
```

```
        i--;
        j++;
        if (i<0)
        {
          i++;
          j--;
          i++;
          j++;
          up=false;
          down=true;
        }
      }
    }

    return ciphertext;
  }
  public static void main(String[] args) {
    String plaintext,ciphertext;
    int key;
    System.out.println("\nRAIL FENCE CIPHER\n\nENCRYPTION\nEnter plaintext:
");
    Scanner s = new Scanner(System.in);
    plaintext = s.nextLine();
    System.out.println("Enter key:");
    key = s.nextInt();
    s.nextLine();
    int pl = plaintext.length();
    ciphertext = encrypt(plaintext,key);
    System.out.println("Ciphertext: "+ciphertext);
    System.out.println("\nDECRYPTION\nEnter ciphertext: ");
    ciphertext = s.nextLine();
    plaintext = decrypt(ciphertext,key);
    System.out.println("Plaintext: "+plaintext);


  }
}
```

ROW COLUMN CIPHER

```
import java.util.*;
class Main {
  static char[][] matrix;
  static String encrypt (String plaintext, String key)
  {
    String ciphertext="";
    int ind=0;
    int kl = key.length();
    int pl = plaintext.length();
    int row;
    if (pl%kl!=0)
      row=(pl/kl)+1;
    else
      row=pl/kl;
    matrix = new char[row][kl];
    for (int i=0;i<row;i++)
    {
      for (int j=0;j<kl;j++)
      {
        if (ind<pl)
          matrix[i][j]=plaintext.charAt(ind++);
        else
          matrix[i][j]='x';
      }
    }
    for (int i=1;i<=kl;i++)
    {
      int c = key.indexOf((char)i+48);
      int r=0;
      for (r=0;r<row;r++)
      {
        ciphertext+=matrix[r][c];
      }
    }
    return ciphertext;
  }
  static String decrypt (String plaintext, String key)
  {
    String ciphertext="";
    int ind=0;
    int kl = key.length();
    int pl = plaintext.length();
    int row;
    if (pl%kl!=0)
```

```
      row=(pl/kl)+1;
    else
      row=pl/kl;
    matrix = new char[row][kl];
    ind=0;
    for (int i=1;i<=kl;i++)
    {
      int c = key.indexOf((char)i+48);
      int r=0;
      for (r=0;r<row;r++)
      {
        matrix[r][c]=plaintext.charAt(ind++);
      }
    }
    for (int i=0;i<row;i++)
    {
      for (int j=0;j<kl;j++)
      {
        ciphertext+=matrix[i][j];
      }
    }

    return ciphertext;
  }
  public static void main(String[] args) {
    String plaintext,ciphertext,key;
    System.out.println("\nROW COLUMN CIPHER\n\nENCRYPTION\nEnter plaintext:
");
    Scanner s = new Scanner(System.in);
    plaintext = s.nextLine();
    System.out.println("Enter key permutation:");
    key = s.nextLine();
    ciphertext = encrypt(plaintext,key);
    System.out.println("Ciphertext: "+ciphertext);
    System.out.println("\nDECRYPTION\nEnter ciphertext:");
    ciphertext = s.nextLine();
    plaintext = decrypt(ciphertext, key);
    System.out.println("Plaintext: "+plaintext);

  }
}
```

DES

```
import java.util.*;
class Main {
  static int[] bkey64 = new int[64];
  static int[][] subkeys = new int[16][48];
  static int[] PC1 = { 57, 49, 41, 33, 25,
                       17, 9, 1, 58, 50, 42, 34, 26,
                       18, 10, 2, 59, 51, 43, 35, 27,
                       19, 11, 3, 60, 52, 44, 36, 63,
                       55, 47, 39, 31, 23, 15, 7, 62,
                       54, 46, 38, 30, 22, 14, 6, 61,
                       53, 45, 37, 29, 21, 13, 5, 28,
                       20, 12, 4 };
    static int[] shiftbits = { 1, 1, 2, 2, 2, 2, 2, 2,
                               1, 2, 2, 2, 2, 2, 2, 1 };
    static int[] PC2 = { 14, 17, 11, 24, 1, 5, 3,
                       28, 15, 6, 21, 10, 23, 19, 12,
                       4, 26, 8, 16, 7, 27, 20, 13, 2,
                       41, 52, 31, 37, 47, 55, 30, 40,
                       51, 45, 33, 48, 44, 49, 39, 56,
                       34, 53, 46, 42, 50, 36, 29, 32 };
    static int[] EP = { 32, 1, 2, 3, 4, 5, 4,
                      5, 6, 7, 8, 9, 8, 9, 10,
                      11, 12, 13, 12, 13, 14, 15,
                      16, 17, 16, 17, 18, 19, 20,
                      21, 20, 21, 22, 23, 24, 25,
                      24, 25, 26, 27, 28, 29, 28,
                      29, 30, 31, 32, 1 };
    static int[][][] sbox = {
            { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
              { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
              { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
              { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } },

            { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
              { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
              { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
              { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } },
            { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
              { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
              { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
              { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } },
            { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
              { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
              { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
```

```
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } },
          { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
            { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
            { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
            { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } },
          { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
            { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
            { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
            { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } },
          { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
            { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
            { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
            { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } },
          { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
            { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
            { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
            { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } }
      };
  static int[] P = { 16, 7, 20, 21, 29, 12, 28,
                     17, 1, 15, 23, 26, 5, 18,
                     31, 10, 2, 8, 24, 14, 32,
                     27, 3, 9, 19, 13, 30, 6,
                     22, 11, 4, 25 };
  static int[] IP = { 58, 50, 42, 34, 26, 18,
                      10, 2, 60, 52, 44, 36, 28, 20,
                      12, 4, 62, 54, 46, 38,
                      30, 22, 14, 6, 64, 56,
                      48, 40, 32, 24, 16, 8,
                      57, 49, 41, 33, 25, 17,
                      9, 1, 59, 51, 43, 35, 27,
                      19, 11, 3, 61, 53, 45,
                      37, 29, 21, 13, 5, 63, 55,
                      47, 39, 31, 23, 15, 7 };
  static int[] IIP = { 40, 8, 48, 16, 56, 24, 64,
                       32, 39, 7, 47, 15, 55,
                       23, 63, 31, 38, 6, 46,
                       14, 54, 22, 62, 30, 37,
                       5, 45, 13, 53, 21, 61,
                       29, 36, 4, 44, 12, 52,
                       20, 60, 28, 35, 3, 43,
                       11, 51, 19, 59, 27, 34,
                       2, 42, 10, 50, 18, 58,
                       26, 33, 1, 41, 9, 49,
                       17, 57, 25 };
  static boolean checkKeywordSize(String keyword) //Checks if key is 8
characters
  {
    if (keyword.length()!=8)
```

```
      return false;
    else= ""
      return true;
  }
  static String padPlaintext (String plaintext) //Pads plaintext to multiple
of 8
  {
    int pl = plaintext.length();
    while (pl%8 != 0)
    {
      plaintext+='x';
      pl = plaintext.length();
    }
    return plaintext;
  }
  static String convertAsciiToBinary (String keyword, String type)
  {
    int ind=0,ind1=0;
    int[] binarray = new int[64];
    for (int i=0;i<keyword.length();i++)
    {
      char c = keyword.charAt(i);
      int ascii = (int)c;
      String binary = Integer.toBinaryString(ascii);
      int binlen = binary.length();
      binlen--;
      int[] bin = new int[8];
      for (int in=7;in>=0;in--)
      {
        if (binlen>=0)
          bin[in]=((int)binary.charAt(binlen--))-48;
        else
          bin[in]=0;
      }
      if (type.equals("key"))
        for (int in=0;in<8;in++)
          bkey64[ind++]=bin[in];
      else
        for (int in=0;in<8;in++)
          binarray[ind1++]=bin[in];
    }
    String result="";
    if (type.equals("key"))
      for (int in=0;in<64;in++)
      {
        char c = (char)(bkey64[in]+48);
        result+=c;
      }
```

```
  else
    for (int in=0;in<64;in++)
    {
      char c = (char)(binarray[in]+48);
      result+=c;
    }
  return result;
}
static String binToHex (String input)
{
  int n = (int)input.length()/4;
  String output = Long.toHexString(Long.parseUnsignedLong(input, 2));
  while (output.length()<n)
    output = "0"+output;
  return output;
}
static String hexToBin (String input)
{
  int n = (int)input.length()*4;
  String output = Long.toBinaryString(Long.parseUnsignedLong(input, 16));
  while (output.length()<n)
    output = "0"+output;
  return output;
}
static void generateSubKeys()
{
  int[] bkey56 = new int[56];
  int[] bkey48 = new int[48];
  int[] bkeyleft28 = new int[28];
  int[] bkeyright28 = new int[28];
  bkey56 = doPC1(bkey64);
  for (int round=1;round<=16;round++)
  {
    for (int i=0;i<28;i++)
      bkeyleft28[i] = bkey56[i];
    for (int i=0;i<28;i++)
      bkeyright28[i] = bkey56[i+28];
    bkeyleft28 = doLeftShift(bkeyleft28,round);
    bkeyright28 = doLeftShift(bkeyright28,round);
    for (int i=0;i<28;i++)
      bkey56[i] = bkeyleft28[i];
    for (int i=0;i<28;i++)
      bkey56[i+28] = bkeyright28[i];
    bkey48 = doPC2(bkey56);
    for (int i=0;i<48;i++)
      subkeys[round-1][i] = bkey48[i];
  }
}
```

```
static int[] doPC1 (int[] bkey64)
{
  int[] bkey56 = new int[56];
  for (int i=0;i<56;i++)
    bkey56[i] = bkey64[PC1[i]-1];
  return bkey56;
}
static int[] doPC2 (int[] bkey56)
{
  int[] bkey48 = new int[48];
  for (int i=0;i<48;i++)
    bkey48[i] = bkey56[PC2[i]-1];
  return bkey48;
}
static int[] doLeftShift (int[] bkey28, int round)
{
  int shift = shiftbits[round-1];
  for (int i=0;i<shift;i++)
  {
    int firstbit = bkey28[0];
    for (int j=1;j<28;j++)
      bkey28[j-1] = bkey28[j];
    bkey28[27] = firstbit;
  }
  return bkey28;
}
static int[] doIP (int[] p64)
{
  int[] result64 = new int[64];
  for (int i=0;i<64;i++)
    result64[i] = p64[IP[i]-1];
  return result64;
}
static int[] doIIP (int[] p64)
{
  int[] result64 = new int[64];
  for (int i=0;i<64;i++)
    result64[i] = p64[IIP[i]-1];
  return result64;
}
static int[] doEP (int[] p32)
{
  int[] p48 = new int[48];
  for (int i=0;i<48;i++)
    p48[i] = p32[EP[i]-1];
  return p48;
}
static int[] doP (int[] p32)
```

```
  {
    int[] r = new int[32];
    for (int i=0;i<32;i++)
      r[i] = p32[P[i]-1];
    return r;
  }
  static int[] xor (int[] a, int[] b, int bits)
  {
    int[] r = new int[bits];
    for (int i=0;i<bits;i++)
    {
      if (a[i]==b[i])
        r[i]=0;
      else
        r[i]=1;
    }
    return r;
  }
  static int[] doSbox (int[] p48)
  {
    int[] p32 = new int[32];
    String plain32="";
    int index=0;
    for (int i=0;i<48;i+=6)
    {
      int sboxrow =
Integer.parseInt(Integer.toString(p48[i])+Integer.toString(p48[i+5]), 2);
      int sboxcol =
Integer.parseInt(Integer.toString(p48[i+1])+Integer.toString(p48[i+2])+Integer
.toString(p48[i+3])+Integer.toString(p48[i+4]), 2);
      int sboxval = sbox[index][sboxrow][sboxcol];
      String binval = Integer.toBinaryString(sboxval);
      while (binval.length()<4)
        binval = "0"+binval;
      plain32+=binval;
      index++;
    }
    for (int i=0;i<32;i++)
      p32[i] = ((int)plain32.charAt(i))-48;
    return p32;
  }
  static String encrypt (String plaintextbin)
  {
    int[] p64 = new int[64];
    int[] c64 = new int[64];
    int[] left32 = new int[32];
    int[] right32 = new int[32];
    int[] right48 = new int[48];
```

```
  int[] newright32 = new int[32];
  int[] xorright32 = new int[32];
  for (int i=0;i<64;i++)
    p64[i] = ((int)plaintextbin.charAt(i))-48;
  p64 = doIP(p64);
  for (int round=1;round<=16;round++)
  {
    for (int i=0;i<32;i++)
      left32[i] = p64[i];
    for (int i=0;i<32;i++)
      right32[i] = p64[i+32];
    right48 = doEP(right32);
    right48 = xor(right48,subkeys[round-1],48);
    newright32 = doSbox(right48);
    newright32 = doP(newright32);
    xorright32 = xor(newright32,left32,32);
    left32 = right32;
    right32 = xorright32;
    for (int i=0;i<32;i++)
      p64[i] = left32[i];
    for (int i=0;i<32;i++)
      p64[i+32] = right32[i];
  }
  for (int i=0;i<32;i++)
    c64[i] = p64[i+32];
  for (int i=32;i<64;i++)
    c64[i] = p64[i-32];
  c64 = doIIP(c64);
  String result="";
  for (int i=0;i<64;i++)
  {
    char c = (char)(c64[i]+48);
    result+=c;
  }
  return result;
}
static String decrypt (String plaintextbin)
{
  int[] p64 = new int[64];
  int[] c64 = new int[64];
  int[] left32 = new int[32];
  int[] right32 = new int[32];
  int[] right48 = new int[48];
  int[] newright32 = new int[32];
  int[] xorright32 = new int[32];
  for (int i=0;i<64;i++)
    p64[i] = ((int)plaintextbin.charAt(i))-48;
  p64 = doIP(p64);
```

```
  for (int round=16;round>=1;round--)
  {
    for (int i=0;i<32;i++)
      left32[i] = p64[i];
    for (int i=0;i<32;i++)
      right32[i] = p64[i+32];
    right48 = doEP(right32);
    right48 = xor(right48,subkeys[round-1],48);
    newright32 = doSbox(right48);
    newright32 = doP(newright32);
    xorright32 = xor(newright32,left32,32);
    left32 = right32;
    right32 = xorright32;
    for (int i=0;i<32;i++)
      p64[i] = left32[i];
    for (int i=0;i<32;i++)
      p64[i+32] = right32[i];
  }
  for (int i=0;i<32;i++)
    c64[i] = p64[i+32];
  for (int i=32;i<64;i++)
    c64[i] = p64[i-32];
  c64 = doIIP(c64);
  String result="";
  for (int i=0;i<64;i++)
  {
    char c = (char)(c64[i]+48);
    result+=c;
  }
  return result;
}
static String convertBinaryToAscii (String text)
{
  String result="";
  for (int i=0;i<text.length();i+=8)
  {
    String block = text.substring(i,i+8);
    int ascii = Integer.parseInt(block,2);
    char c = (char)ascii;
    result+=c;
  }
  return result;
}
public static void main(String[] args) {
  String plaintext,cipherhex,keyword;
  String keybinary,keyhex;
  System.out.println("\nDES\n\nEnter plaintext (in ASCII): ");
  Scanner s = new Scanner(System.in);
```

```
plaintext = s.nextLine();
System.out.println("Enter keyword (in ASCII-only 8 characters): ");
keyword = s.nextLine();
if (!checkKeywordSize(keyword))
{
  System.out.println("Invalid keyword");
  System.exit(0);
}
plaintext = padPlaintext(plaintext);
keybinary = convertAsciiToBinary(keyword,"key");
keyhex = binToHex(keybinary);
System.out.println("\nKey in hex: "+keyhex);
generateSubKeys();
System.out.println("\nKEY GENERATION");
for (int i=0;i<16;i++)
{
  String rkey="";
  for (int j=0;j<48;j++)
  {
    char c = (char)(subkeys[i][j]+48);
    rkey+=c;
  }
  String rkeyhex = binToHex(rkey);
  System.out.println("Round "+(i+1)+" subkey: "+rkeyhex);
}
System.out.println("\n\nENCRYPTION");
cipherhex="";
String plaintexthex="";
for (int i=0;i<plaintext.length();)
{
  String plainblock = plaintext.substring(i,i+8);
  String plainblockbin = convertAsciiToBinary(plainblock,"plain");
  plaintexthex+=binToHex(plainblockbin);
  String cipherblockbin = encrypt(plainblockbin);
  String cipherblockhex = binToHex(cipherblockbin);
  cipherhex+=cipherblockhex;
  i+=8;
}
System.out.println("\n\nPlaintext in hex: "+plaintexthex);
System.out.println("Ciphertext in hex: "+cipherhex);

System.out.println("\n\nDECRYPTION\nEnter ciphertext in hex: ");
cipherhex = s.nextLine();
plaintexthex="";
plaintext="";
for (int i=0;i<cipherhex.length();)
{
  String cipherblockhex = cipherhex.substring(i,i+16);
```

```
      String cipherblockbin = hexToBin(cipherblockhex);
      String plainblockbin = decrypt(cipherblockbin);
      plaintext+=convertBinaryToAscii(plainblockbin);
      plaintexthex+=binToHex(plainblockbin);
      i+=16;
    }
    System.out.println("\n\nPlaintext in hex: "+plaintexthex);
    System.out.println("Plaintext: "+plaintext);




  }
}
```

AES

```
import java.security.MessageDigest;
import java.util.*;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class aes {

   static SecretKeySpec secretKey;
   static byte[] keyarray;
   static void generateKey(String keystring)
   {
      MessageDigest md5;
      try {
         keyarray = keystring.getBytes("UTF-8");
         md5 = MessageDigest.getInstance("MD5");
         keyarray = md5.digest(keyarray);
         keyarray = Arrays.copyOf(keyarray, 16);
         secretKey = new SecretKeySpec(keyarray, "AES");
      }
      catch (Exception e) {
         System.out.println(e);
      }
   }
```

```
   static String encrypt(String plaintext, String keystring)
   {
      try
      {
         generateKey(keystring);
         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
         cipher.init(Cipher.ENCRYPT_MODE, secretKey);
         return
Base64.getEncoder().encodeToString(cipher.doFinal(plaintext.getBytes("UTF-
8")));
      }
      catch (Exception e) {
         System.out.println(e);
      }
      return null;
   }

   static String decrypt(String ciphertext, String keystring)
   {
      try
      {
         generateKey(keystring);
         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
         cipher.init(Cipher.DECRYPT_MODE, secretKey);
         return new
String(cipher.doFinal(Base64.getDecoder().decode(ciphertext)));
      }
```

```
        catch (Exception e) {
            System.out.println(e);
        }
        return null;
    }
    public static void main(String[] args)
    {
        String keystring, plaintext, ciphertext;
        Scanner s = new Scanner(System.in);
        System.out.println("\n\n\t\tADVANCED ENCRYPTION STANDARD (AES)");
        System.out.println("\t\t================================");
        System.out.println("\nEnter key: ");
        keystring = s.nextLine();
        System.out.println("\n\t\tKEY GENERATION\n\t\t==============");
        generateKey(keystring);
        System.out.println("\nMD5 hash of key (in Base64 format):
"+Base64.getEncoder().encodeToString(keyarray));
        System.out.println("\n\n\t\tENCRYPTION\n\t\t==========");
        System.out.println("\nEnter plaintext: ");
        plaintext = s.nextLine();
        ciphertext = encrypt(plaintext, keystring);
        System.out.println("\nCiphertext (in Base64 format): "+ciphertext);
        System.out.println("\n\n\t\tDECRYPTION\n\t\t==========");
        System.out.println("\nEnter ciphertext (in Base64 format): ");
        ciphertext = s.nextLine();
        plaintext = decrypt(ciphertext, keystring);
        System.out.println("\nPlaintext is: "+plaintext);
```

```
    }
}
```

AES KEY GENERATION

```
import java.util.*;
class Main {
  static String[] roundkeys = new String[10];
  static String[] rcon = {"01","02","04","08","10","20","40","80","1b","36"};
  static int[][] sbox = {{0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76},
        {0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0},
        {0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15},
        {0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75},
        {0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84},
        {0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf},
        {0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8},
        {0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2},
        {0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73},
        {0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb},
        {0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79},
        {0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08},
        {0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a},
        {0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e},
        {0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf},
        {0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16}};
  static String leftCircularShift (String input)
  {
    String temp = input.substring(2);
    String result = "";
    result= temp+input.charAt(0)+input.charAt(1);
    return result;
  }
  static String doSbox (String input)
```

```
  {
    String res="";
    for (int i=0;i<input.length();i+=2)
    {
      int r=Integer.parseInt(Character.toString(input.charAt(i)),16);
      int c=Integer.parseInt(Character.toString(input.charAt(i+1)),16);
      int val = sbox[r][c];
      String temp = Integer.toHexString(val);
      while (temp.length()<2)
        temp = "0"+temp;
      res+=temp;
    }

    return res;
  }
  static String hexToBin (String input)
  {
    int n = (int)input.length()*4;
    String output = Long.toBinaryString(Long.parseUnsignedLong(input, 16));
    while (output.length()<n)
      output = "0"+output;
    return output;
  }
  static String binToHex (String input)
  {
    int n = (int)input.length()/4;
    String output = Long.toHexString(Long.parseUnsignedLong(input, 2));
    while (output.length()<n)
      output = "0"+output;
    return output;
  }
  static String xorRcon(String a, String b) //For xor-ing first two bits of a
with two bits of b
  {
    String temp = a.substring(2);
    String a1 = a.substring(0,2);
    a1 = hexToBin(a1);
    b = hexToBin(b);
    String res="";
    for (int i=0;i<8;i++)
    {
      char c1 = a1.charAt(i);
      char c2 = b.charAt(i);
      if (c1==c2)
        res+="0";
      else
        res+="1";
    }
```

```
  res=binToHex(res);
  return res+temp;
}
static String xor(String a, String b)//For xor-ing two 32-bit values
{
  a = hexToBin(a);
  b = hexToBin(b);
  String res="";
  for (int i=0;i<32;i++)
  {
    char c1 = a.charAt(i);
    char c2 = b.charAt(i);
    if (c1==c2)
      res+="0";
    else
      res+="1";
  }
  res=binToHex(res);
  return res;
}
public static void main(String[] args) {
  String keyhex;
  System.out.println("\n\nAES KEY GENERATION\nEnter 128-bit key in hex: ");
  Scanner s = new Scanner(System.in);
  keyhex = s.nextLine();
  for (int round=1;round<=10;round++)
  {
    String w0 = keyhex.substring(0,8);
    String w1 = keyhex.substring(8,16);
    String w2 = keyhex.substring(16,24);
    String w3 = keyhex.substring(24);
    String lw3 = leftCircularShift(w3);
    String sw3 = doSbox(lw3);
    String z1 = xorRcon(sw3,rcon[round-1]);
    String w4 = xor(w0,z1);
    String w5 = xor(w1,w4);
    String w6 = xor(w2,w5);
    String w7 = xor(w3,w6);
    keyhex = w4+w5+w6+w7;
    roundkeys[round-1]=keyhex;
  }
  System.out.println("\nROUND KEYS");
  for (int i=0;i<10;i++)
  {
    System.out.println("ROUND "+(i+1)+" KEY: "+roundkeys[i]);
  }

}
```

```
}
```

RSA

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.*;

public class rsa
{
   static BigInteger p,q,n;
   static BigInteger phi;
   static BigInteger e;
   static BigInteger d;
   static int num_of_bits = 1024;
   static Random r;

   static void generateKeys()
   {
      r = new Random();
      p = BigInteger.probablePrime(num_of_bits, r);
      q = BigInteger.probablePrime(num_of_bits, r);
      n = p.multiply(q);
      phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
      e = BigInteger.probablePrime(num_of_bits/2, r);
      while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) <
0)
```

```
        {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }
    public static void main(String[] args) throws IOException
    {
        String plaintext,ciphertext;
        Scanner s = new Scanner(System.in);
        System.out.println("\n\n\t\tRSA ALGORITHM (1024
bits)\n\t\t=========================");
        System.out.println("\nKEY GENERATION\n==============");
        generateKeys();
        System.out.println("\nPublic key:\n\ne:\n\n"+e+"\n\nn:\n\n"+n);
        System.out.println("\nPrivate key:\n\nd:\n\n"+d);
        System.out.println("\nENCRYPTION\n==========");
        System.out.println("\nEnter plaintext:");
        plaintext = s.nextLine();
        System.out.println("\nPlaintext (in byte format): \n"+
convertByteArrayToString(plaintext.getBytes()));
        BigInteger ciphertxt = encrypt(plaintext);
        System.out.println("\nCiphertext: \n\n"+ciphertxt);
        System.out.println("\n\nDECRYPTION\n==========");
        System.out.println("\nEnter ciphertext:\n");
        ciphertext = s.nextLine();
        plaintext = decrypt(ciphertext);
        System.out.println("\nPlaintext is: "+plaintext);
```

```
    }

    static String convertByteArrayToString(byte[] encrypted)
    {
        String test = "";
        for (byte b : encrypted)
        {
            test += Byte.toString(b);
        }
        return test;
    }
    static BigInteger encrypt(String plaintext)
    {
        byte[] plainbytes = plaintext.getBytes();
        BigInteger plainint = new BigInteger(plainbytes);
        BigInteger result = plainint.modPow(e, n);
        return result;
    }
    static String decrypt(String ciphertext)
    {
        //byte[] cipherbytes = Base64.getDecoder().decode(ciphertext);
        BigInteger cipherint = new BigInteger(ciphertext);
        BigInteger result = cipherint.modPow(d, n);
        return new String(result.toByteArray());
    }
}
```

DHKE

```
import java.io.IOException;
import java.math.*;
import java.util.*;
public class DiffieHellmanKeyExchange {
 BigInteger p;
 BigInteger g;
 BigInteger phi;
 BigInteger Xa;
 BigInteger Xb;
 BigInteger Ya;
 BigInteger Yb;
 BigInteger Ka;
 BigInteger Kb;
 int bitlength = 32;
 int noOfIterations = 5;
 private Random random;
 static BigInteger ZERO = BigInteger.ZERO;
 static BigInteger ONE = BigInteger.ONE;
 static BigInteger big2 = new BigInteger("2");
 static BigInteger big3 = new BigInteger("3");
 static BigInteger big4 = new BigInteger("4");
 public DiffieHellmanKeyExchange() {
 random = new Random();
 generatePrime();
```

```
generatePrimitiveRoot();
}
public static BigInteger sqrt(BigInteger val) {
  BigInteger half = BigInteger.ZERO.setBit(val.bitLength() / 2);
  BigInteger cur = half;

  while (true) {
    BigInteger tmp = half.add(val.divide(half)).shiftRight(1);

    if (tmp.equals(half) || tmp.equals(cur))
      return tmp;

    cur = half;
    half = tmp;
  }
}
public void generatePrimitiveRoot() {
phi = p.subtract(ONE);
HashSet<BigInteger> primeFactors = getPrimeFactors();
ArrayList<BigInteger> primitiveRoots = new ArrayList<>();
for (BigInteger r = big2;r.compareTo(phi) < 0;r = r.add(BigInteger.ONE)) {
boolean flag = false;
for (BigInteger l : primeFactors) {
BigInteger phiBig = phi.divide(l);
BigInteger pr = r.modPow(phiBig, p);
if (pr.compareTo(BigInteger.valueOf(1)) == 0) {
```

```
flag = true;
break;
}
}
if (!flag) {
primitiveRoots.add(r);
}
}
g= primitiveRoots.get(new Random().nextInt(primitiveRoots.size()));
}
public HashSet<BigInteger> getPrimeFactors() {
HashSet<BigInteger> primesFactors = new HashSet<>();
while (phi.mod(big2).signum() == 0) {
primesFactors.add(big2);
phi = phi.divide(big2);
}
for (BigInteger i = big3; i.compareTo(sqrt(phi)) <= 0; i = i.add(big2)) {
if (phi.mod(i).signum() == 0) {
primesFactors.add(i);
phi = phi.divide(i);
}
}
if (phi.compareTo(big2) > 0) {
primesFactors.add(phi);
}
return primesFactors;
```

```
}
void generatePrime() {
byte[] b = new byte[bitlength / 8];
random.nextBytes(b);
p = new BigInteger(b);
while (!isPrime(p, noOfIterations)) {
random.nextBytes(b);
p = new BigInteger(b);
}
}
boolean millerRabinCheck(BigInteger d, BigInteger n) {
BigInteger maxLimit = n.subtract(big2);
BigInteger minLimit = big2;
BigInteger bigInteger = maxLimit.subtract(minLimit);
int len = maxLimit.bitLength();
BigInteger a = new BigInteger(len, random);
if (a.compareTo(minLimit) < 0) a = a.add(minLimit);
if (a.compareTo(bigInteger) >= 0) a = a.mod(bigInteger).add(minLimit);
BigInteger x = a.modPow(d, n);
if (x.compareTo(ONE) == 0 || x.compareTo(n.subtract(ONE)) == 0) return true;
while (d.compareTo(n.subtract(ONE)) != 0) {
x = x.multiply(x).mod(n);
d = d.multiply(big2);
if (x.compareTo(ONE) == 0) return false;
if (x.compareTo(n.subtract(ONE)) == 0) return true;
}
```

```
return false;
}
boolean isPrime(BigInteger n, int k) {

if (n.compareTo(ONE) <= 0 || n.compareTo(big4) == 0) return false;
if (n.compareTo(big3) <= 0) return true;
BigInteger d = n.subtract(ONE);
while (d.mod(big2).signum() == 0) d = d.divide(big2);
for (int i = 0; i < k; i++) if (!millerRabinCheck(d, n)) return false;
return true;
}
void keyGenA(){
BigInteger maxLimit = g;
BigInteger minLimit = ONE;
BigInteger bigInteger = maxLimit.subtract(minLimit);
int len = maxLimit.bitLength();
Xa = new BigInteger(len, random);
if (Xa.compareTo(minLimit) < 0)
Xa = Xa.add(minLimit);
if (Xa.compareTo(bigInteger) >= 0)
Xa = Xa.mod(bigInteger).add(minLimit);
Ya=g.modPow(Xa,p);
}
void keyGenB(){
BigInteger maxLimit = g;
BigInteger minLimit = ONE;
```

```
BigInteger bigInteger = maxLimit.subtract(minLimit);
int len = maxLimit.bitLength();
Xb = new BigInteger(len, random);
if (Xb.compareTo(minLimit) < 0)
Xb = Xb.add(minLimit);
if (Xb.compareTo(bigInteger) >= 0)
Xb = Xb.mod(bigInteger).add(minLimit);
Yb=g.modPow(Xb,p);
}
void sharedKeyA(){
Ka=Yb.modPow(Xa,p);
}
void sharedKeyB(){
Kb=Ya.modPow(Xb,p);
}
public static void main(String[] args) throws IOException {
Scanner sc = new Scanner(System.in);
System.out.println("\n\n\t\tDIFFIE HELLMAN KEY EXCHANGE ALGORITHM");
System.out.println("\t\t====================================");
DiffieHellmanKeyExchange dhke = new DiffieHellmanKeyExchange();
System.out.println("\n\nPrime and primitive root generation");
System.out.println("==================================");
System.out.println("\nPrime number P (Big Integer): "+dhke.p);
System.out.println("\nPrimitive root G (Big Integer): "+ dhke.g);
dhke.keyGenA();
System.out.println("\n\nKEY GENERATION for User A");
```

```
System.out.println("========================");
System.out.println("\nPrivate key of A (Big Integer): "+ dhke.Xa);
System.out.println("\nPublic key of A (Big Integer): "+ dhke.Ya);
dhke.keyGenB();
System.out.println("\n\nKEY GENERATION for User B");
System.out.println("========================");
System.out.println("\nPrivate key of B (Big Integer): "+ dhke.Xb);
System.out.println("\nPublic key of B (Big Integer): "+ dhke.Yb);
dhke.sharedKeyA();
System.out.println("\n\nSECRET KEY CALCULATION");
System.out.println("======================");
System.out.println("\nShared secret key calculated by User A (Big Integer): "+
dhke.Ka);
dhke.sharedKeyB();
System.out.println("\nShared secret key calculated by User B (Big Integer): "+
dhke.Kb);
}
}
```

SHA1 (USING LIBRARY)

```
import java.math.BigInteger;
import java.security.*;
import java.util.*;

public class sha1 {
	public static String sha1Digest(String message)
	{
		String hexHash="";
		try
			{
			MessageDigest md = MessageDigest.getInstance("SHA-1");
			byte[] hash = md.digest(message.getBytes());
			BigInteger number = new BigInteger(1,hash);
			hexHash = number.toString(16);
			}
		catch (Exception e)
		{
			System.out.println("Error! "+e);
		}
		return hexHash;
	}

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

```
                String message,digest;
                Boolean flag=true;
                Scanner s = new Scanner(System.in);
                System.out.println("\n\n\t\tSHA1 ALGORITHM");
                System.out.println("\t\t==============");
                while (flag)
                {
                        System.out.println("\nEnter message:");
                        message = s.nextLine();
                        digest = sha1Digest(message);
                        System.out.println("\nSHA1 message digest (in
Hexadecimal): "+digest);
                        System.out.print("\n\nDO YOU WANT TO TRY AGAIN?(y/n):
");
                        String choice = s.nextLine();
                        if (choice.equals("y"))
                                flag=true;
                        else
                                flag=false;
                }

        }
}
```

SHA1 (WITHOUT LIBRARY)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Main {
	private List<byte[]> inputDataList = new ArrayList<byte[]>();


	// Bitwise rotate a 32-bit number to the left
	private static int rol(int num, int cnt) {
		return (num << cnt) | (num >>> (32 - cnt));
	}


	public byte[] digest(byte[] x) {


		// Append padding bits and the length
		int[] blks = new int[(((x.length + 8) >> 6) + 1) * 16];
		int i;
		for(i = 0; i < x.length; i++) {
			blks[i >> 2] |= x[i] << (24 - (i % 4) * 8);
		}
		blks[i >> 2] |= 0x80 << (24 - (i % 4) * 8);
		blks[blks.length - 1] = x.length * 8;


		// calculate 160 bit SHA1 hash of the sequence of blocks


		int[] w = new int[80];


		int a =  1732584193;
		int b = -271733879;
		int c = -1732584194;
		int d =  271733878;
		int e = -1009589776;


		for(i = 0; i < blks.length; i += 16) {
			int olda = a;
			int oldb = b;
			int oldc = c;
			int oldd = d;
			int olde = e;
```

```
                        for(int j = 0; j < 80; j++) {
                                w[j] = (j < 16) ? blks[i + j] :
                                        ( rol(w[j-3] ^ w[j-8] ^ w[j-14] ^ w[j-
16], 1) );


                                int t = rol(a, 5) + e + w[j] +
                                                ( (j < 20) ?  1518500249 + ((b
& c) | ((~b) & d))
                                                                : (j < 40) ?
1859775393 + (b ^ c ^ d)

: (j < 60) ? -1894007588 + ((b & c) | (b & d) | (c & d))

: -899497514 + (b ^ c ^ d) );
                                e = d;
                                d = c;
                                c = rol(b, 30);
                                b = a;
                                a = t;
                        }


                        a = a + olda;
                        b = b + oldb;
                        c = c + oldc;
                        d = d + oldd;
                        e = e + olde;
                }


                // Convert result to a byte array
                byte[] digest = new byte[20];
                fill(a, digest, 0);
                fill(b, digest, 4);
                fill(c, digest, 8);
                fill(d, digest, 12);
                fill(e, digest, 16);


                return digest;
        }


        private void fill(int value, byte[] arr, int off) {
                arr[off + 0] = (byte) ((value >> 24) & 0xff);
                arr[off + 1] = (byte) ((value >> 16) & 0xff);
```

```
                arr[off + 2] = (byte) ((value >> 8) & 0xff);
                arr[off + 3] = (byte) ((value >> 0) & 0xff);
        }
        private static final char[] HEX_ARRAY =
"0123456789ABCDEF".toCharArray();
        public static String bytesToHex(byte[] bytes) {
            char[] hexChars = new char[bytes.length * 2];
            for (int j = 0; j < bytes.length; j++) {
                int v = bytes[j] & 0xFF;
                hexChars[j * 2] = HEX_ARRAY[v >>> 4];
                hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
            }
            return new String(hexChars);
        }
        public static void main(String[] args) {
                Main Main = new Main();
                System.out.println("Enter the string: ");
                Scanner scanner = new Scanner(System.in);
                String p = scanner.nextLine();
                byte[] digest = Main.digest(p.getBytes());
                System.out.println("The SHA1 hash is "+bytesToHex(digest));
        }
}
```

DSA

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

public class signature
{
    private static KeyPairGenerator keyPairGen;
    private static KeyPair pair;
    private static Signature sign;
    private static String message;
    static String signMessage(String message)
    {
        try
        {
            keyPairGen = KeyPairGenerator.getInstance("DSA");
            keyPairGen.initialize(2048);
            pair = keyPairGen.generateKeyPair();
            PrivateKey privKey = pair.getPrivate();
            sign = Signature.getInstance("SHA256withDSA");
            sign.initSign(privKey);
            byte[] bytes = message.getBytes();
            sign.update(bytes);
            byte[] digisignature = sign.sign();
            return Base64.getEncoder().encodeToString(digisignature);
        }
```

```
    catch(Exception e)
    {
        //System.out.println("Error "+e);
    }
    return null;
}

static boolean verifySign(byte[] digisignature)
{
    try
    {
        byte[] bytes = message.getBytes();
        sign.initVerify(pair.getPublic());
        sign.update(bytes);
        boolean bool = sign.verify(digisignature);
        return bool;
    }
    catch(Exception e)
    {
       // System.out.println("Error "+e);
    }
    return false;
}

public static void main(String args[]) throws Exception
{
```

```
        System.out.println("\n\n\t\tDIGITAL SIGNATURE STANDARD
IMPLEMENTATION");


System.out.println("\t\t======================================");
        System.out.println("\nCREATION OF DIGITAL SIGNATURE");
        System.out.println("============================");
        System.out.println("\nEnter message: ");
        Scanner s = new Scanner(System.in);
        message = s.nextLine();
        String digisignaturestring = signMessage(message);
        System.out.println("Digital signature of message (in Base64
format):\n"+digisignaturestring);
        System.out.println("\n\nVERIFICATION OF DIGITAL SIGNATURE");
        System.out.println("================================");
        System.out.println("\nEnter digital signature (in Base64 format): ");
        String ipsign = s.nextLine();
        boolean result = verifySign(Base64.getDecoder().decode(ipsign));
        if (result)
        {
            System.out.println("\nSignature is verified");
        }
        else
            System.out.println("\nSignature failed");


    }
}
```