MADRAS INSTITUTE OFTECHNOLOGY
ANNA UNIVERSITY

DEPARTMENT OF INFORMATION TECHNOLOGY
IT5711 – MOBILE AND SECURITY LABORATORY

RECORD

REGISTER NUMBER: 2020506087

NAME : SHRI KAANTH P

SEMESTER : VII

DEPARTMENT OF INFORMATION TECHNOLOGY

ANNA UNIVERSITY, MIT CAMPUS

CHROMEPET, CHENNAI – 600 044

BONAFIDE CERTIFICATE

Certified that the bonafide record of the practical work done by Mr./Ms. Shri Kaanth P Register Number (2020506087) of **Seventh** Semester, **B.Tech Information Technology** in the **IT5711-Mobile and Security Laboratory** during the academic period from **August 2023 to December 2023**

Submitted for Practical Examination held on _____

Date:

Course Instructor

Dr. B. Lydia Elizabeth

Internal Examiner

External Examiner

TABLE OF CONTENTS

EXP NO :1.a **DATE:** 04/08/2023	**CAESER CIPHER**

AIM:

To perform encryption and decryption using Caeser cipher.

CODE:

```
import java.io.*;
public class caeser_cipher {

   private static void encrypt(String word,int key)
   {
      String cipher="";
      for(int i=0;i<word.length();i++)
      {
         int ascii;
         if(Character.isUpperCase(word.charAt(i)))   ascii=65;
         else ascii=97;
         int c=((word.charAt(i)+key-ascii)%26)+ascii;
         char m=(char)c;
         cipher+=m;
      }
      System.out.println("Cipher text: " + cipher);
   }

   private static void decrypt(String word,int key)
   {
      String plain="";
      for(int i=0;i<word.length();i++)
      {
         int ascii;
         if(Character.isUpperCase(word.charAt(i)))   ascii=65;
         else ascii=97;
         int c=((word.charAt(i)-key-ascii+26)%26)+ascii;
         char m=(char)c;
         plain+=m;
      }
      System.out.println("Plain text: " + plain);
   }

   public static void main(String[] args) throws IOException
   {
      BufferedReader buf = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Enter key: ");
      int key = Integer.parseInt(buf.readLine());
```

```
        while(true)
        {
            System.out.print("\nEnter a word: ");
            String word = buf.readLine();
            System.out.print("1.Encrypt\t2.Decrypt\t3.Exit\nEnter your choice: ");
            int choice = Integer.parseInt(buf.readLine());
            switch(choice)
            {
                case 1:
                    encrypt(word,key);
                    break;

                case 2:
                    decrypt(word,key);
                    break;

                case 3:
                    System.exit(1);
            }
        }
    }
}
```

OUTPUT:

```
Enter key: 5

Enter a word: HeLlOwOrLd
1.Encrypt       2.Decrypt       3.Exit
Enter your choice: 1
Cipher text: MjQqTbTwQi

Enter a word: MjQqTbTwQi
1.Encrypt       2.Decrypt       3.Exit
Enter your choice: 2
Plain text: HeLlOwOrLd
```

RESULT:

Thus, encryption and decryption using Caeser Cipher has been performed successfully and the output is verified.

EXP NO :1.b **DATE:** 04/08/2023	**AFFINE CIPHER**

AIM:

To perform encryption and decryption using Affine cipher.

CODE:

```
import java.util.Scanner;
public class affine_cipher {
   static int a,b;
   private static void encrypt(String word)
   {
      String s2="";
      for(int i=0;i<word.length();i++)
      {
         if(word.charAt(i)!=' ')
         {
            int ascii;
            if(Character.isUpperCase(word.charAt(i)))   ascii=65;
            else ascii=97;
            int x=word.charAt(i) - ascii;
            int e = (a*x+b)%26;
            e+=ascii;
            char y=(char)e;
            s2+=y;
         }
         else{
            s2+=word.charAt(i);
         }
      }
      System.out.println("Cipher text: "+s2);

   }

   public static void main(String []args)
   {
      Scanner s=new Scanner(System.in);
      System.out.print("Enter key values a and b: ");
      a=s.nextInt();
      b=s.nextInt();
      while(true)
      {
         String s1="";
         System.out.print("\nEnter a word: ");
         s1=s.next();
```

```
            System.out.print("\n1.Encrypt\t2.Decrypt\t3.Exit");
            System.out.print("\nEnter your choice: ");
            int ch=s.nextInt();
            switch(ch)
            {
                case 1:
                    encrypt(s1);
                    break;

                case 2:
                    decrypt(s1);
                    break;

                    case 3:
                        System.exit(0);
            }
        }
    }

    private static int inverse(int a, int m) {
        a = a % m;
        for (int x = 1; x < m; x++)
            if ((a * x) % m == 1)
                return x;
        return 1;
    }

    private static void decrypt(String word) {
        int aInverse = inverse(a, 26);
        String plain="";

        for (int i = 0; i < word.length(); i++) {

            if (word.charAt(i)!=' ') {
                char ascii = Character.isLowerCase(word.charAt(i)) ? 'a' : 'A';
                int decryptedChar = (aInverse * (word.charAt(i) - ascii - b + 26)) % 26 + ascii;
                plain+=((char) decryptedChar);
            } else {
                plain+=word.charAt(i);
            }
        }

        System.out.println("Plain text: "+plain);
    }
}
```

OUTPUT:

```
Enter key values a and b: 17 20

Enter a word: HeLlOwOrLd

1.Encrypt       2.Decrypt       3.Exit
Enter your choice: 1
Cipher text: JkZzYeYxZt

Enter a word: JkZzYeYxZt

1.Encrypt       2.Decrypt       3.Exit
Enter your choice: 2
Plain text: HeLlOwOrLd
```

RESULT:

Thus, encryption and decryption using Affine Cipher has been performed successfully and the output is verified.

EXP NO :1.c **DATE:** 11/08/2023	**HILL CIPHER**

AIM:

To perform encryption and decryption using Hill cipher.

CODE:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class hill_cipher
{
   int keymatrix[][];
   int linematrix[];
   int resultmatrix[];

   public void divide(String temp, int s)
   {
      while (temp.length() > s)
      {
         String sub = temp.substring(0, s);
         temp = temp.substring(s, temp.length());
         perform(sub);
      }
      if (temp.length() == s)
         perform(temp);
      else if (temp.length() < s)
      {
         for (int i = temp.length(); i < s; i++)
            temp = temp + 'x';
         perform(temp);
      }
   }

   public void perform(String line)
   {
      linetomatrix(line);
      linemultiplykey(line.length());
      result(line.length());
   }

   public void keytomatrix(String key, int len)
   {
      keymatrix = new int[len][len];
      int c = 0;
```

```
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            keymatrix[i][j] = ((int) key.charAt(c)) - 97;
            c++;
        }
    }
}

public void linetomatrix(String line)
{
    linematrix = new int[line.length()];
    for (int i = 0; i < line.length(); i++)
    {
        linematrix[i] = ((int) line.charAt(i)) - 97;
    }
}

public void linemultiplykey(int len)
{
    resultmatrix = new int[len];
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            resultmatrix[i] += keymatrix[i][j] * linematrix[j];
        }
        resultmatrix[i] %= 26;
    }
}

public void result(int len)
{
    String result = "";
    for (int i = 0; i < len; i++)
    {
        result += (char) (resultmatrix[i] + 97);
    }
    System.out.print(result);
}

public boolean check(String key, int len)
{
    keytomatrix(key, len);
    int d = determinant(keymatrix, len);
    d = d % 26;
    if (d == 0)
```

```
        {
            System.out
                    .println("Invalid key!!! Key is not invertible because determinant=0...");
            return false;
        }
        else if (d % 2 == 0 || d % 13 == 0)
        {
            System.out
                    .println("Invalid key!!! Key is not invertible because determinant has common
factor with 26...");
            return false;
        }
        else
        {
            return true;
        }
    }

    public int determinant(int A[][], int N)
    {
        int res;
        if (N == 1)
            res = A[0][0];
        else if (N == 2)
        {
            res = A[0][0] * A[1][1] - A[1][0] * A[0][1];
        }
        else
        {
            res = 0;
            for (int j1 = 0; j1 < N; j1++)
            {
                int m[][] = new int[N - 1][N - 1];
                for (int i = 1; i < N; i++)
                {
                    int j2 = 0;
                    for (int j = 0; j < N; j++)
                    {
                        if (j == j1)
                            continue;
                        m[i - 1][j2] = A[i][j];
                        j2++;
                    }
                }
                res += Math.pow(-1.0, 1.0 + j1 + 1.0) * A[0][j1]
                        * determinant(m, N - 1);
            }
        }
```

```
    return res;
}

public void cofact(int num[][], int f)
{
    int b[][], fac[][];
    b = new int[f][f];
    fac = new int[f][f];
    int p, q, m, n, i, j;
    for (q = 0; q < f; q++)
    {
        for (p = 0; p < f; p++)
        {
            m = 0;
            n = 0;
            for (i = 0; i < f; i++)
            {
                for (j = 0; j < f; j++)
                {
                    b[i][j] = 0;
                    if (i != q && j != p)
                    {
                        b[m][n] = num[i][j];
                        if (n < (f - 2))
                            n++;
                        else
                        {
                            n = 0;
                            m++;
                        }
                    }
                }
            }
            fac[q][p] = (int) Math.pow(-1, q + p) * determinant(b, f - 1);
        }
    }
    trans(fac, f);
}

void trans(int fac[][], int r)
{
    int i, j;
    int b[][], inv[][];
    b = new int[r][r];
    inv = new int[r][r];
    int d = determinant(keymatrix, r);
    int mi = mi(d % 26);
    mi %= 26;
```

```
    if (mi < 0)
        mi += 26;
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < r; j++)
        {
            b[i][j] = fac[j][i];
        }
    }
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < r; j++)
        {
            inv[i][j] = b[i][j] % 26;
            if (inv[i][j] < 0)
                inv[i][j] += 26;
            inv[i][j] *= mi;
            inv[i][j] %= 26;
        }
    }
    System.out.print("\nInverse key:");
    matrixtoinvkey(inv, r);
}

public int mi(int d)
{
    int q, r1, r2, r, t1, t2, t;
    r1 = 26;
    r2 = d;
    t1 = 0;
    t2 = 1;
    while (r1 != 1 && r2 != 0)
    {
        q = r1 / r2;
        r = r1 % r2;
        t = t1 - (t2 * q);
        r1 = r2;
        r2 = r;
        t1 = t2;
        t2 = t;
    }
    return (t1 + t2);
}

public void matrixtoinvkey(int inv[][], int n)
{
    String invkey = "";
    for (int i = 0; i < n; i++)
```

```
        {
            for (int j = 0; j < n; j++)
            {
                invkey += (char) (inv[i][j] + 97);
            }
        }
        System.out.print(invkey);
    }

    public static void main(String args[]) throws IOException
    {
        HillCipher obj = new HillCipher();
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        int choice;
        System.out.print("Enter plain/cipher text: ");
        String line = in.readLine();
        System.out.print("Enter the key/inverse key: ");
        String key = in.readLine();
        double sq = Math.sqrt(key.length());
        if (sq != (long) sq)
            System.out
                    .println("Invalid key length!!! Does not form a square matrix...");
        else
        {

            System.out.print("1: Encryption\t2: Decryption\nSelect your choice: ");
            choice = Integer.parseInt(in.readLine());
            int s = (int) sq;
            if (obj.check(key, s))
            {
                System.out.print("Result:");
                obj.divide(line, s);
                obj.cofact(obj.keymatrix, s);
            }
        }
    }
}
```

OUTPUT:

```
Enter plain/cipher text: text
Enter the key/inverse key: ddcf
1: Encryption   2: Decryption
Select your choice: 1
Result:rgwl
Inverse key:pruj

Enter plain/cipher text: rgwl
Enter the key/inverse key: pruj
1: Encryption   2: Decryption
Select your choice: 2
Result:text
Inverse key:ddcf
```

RESULT:

Thus, encryption and decryption using Hill Cipher has been performed successfully and the output is verified.

EXP NO :1.d **DATE:** 11/08/2023	**TRANSPOSITION CIPHER**

AIM:

To perform encryption and decryption using Transposition cipher.

CODE:

```
import java.util.Scanner;
public class transposition_cipher {

   static void encrypt(String word)
   {
      int l=word.length();
      String s1="",s2="",res="";
      for(int i=0;i<l/2;i++)
         s1+=word.charAt(i);
      for(int i=l/2;i<l;i++)
         s2+=word.charAt(i);

      if(l%2==0)
      {
         for(int i=0;i<l/2;i++)
         {
            res+=s1.charAt(i);
            res+=s2.charAt(l/2-i-1);
         }
      }

      if(l%2==1)
      {
         for(int i=0;i<l/2;i++)
         {
            res+=s1.charAt(i);
            res+=s2.charAt(l/2-i);
         }
         res+=s2.charAt(0);
      }

      System.out.println("Cipher Text: "+res);
   }

   static void decrypt(String word)
   {
      String s1="",s2="",res="";
      int l=word.length();
```

```
        for(int i=0;i<l/2;i++)
        {
            s1+=word.charAt(2*i);
            s2+=word.charAt(2*i+1);
        }

        if(l%2==1)
            s2+=word.charAt(l-1);
        res+=s1;
        for(int i=0;i<s2.length();i++)
        {
            res=res+s2.charAt(s2.length()-i-1);
        }
        System.out.println("Plain text: "+res);
    }

    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);
        while(true)
        {
            System.out.print("\nEnter a word: ");
            String word = s.next();
            System.out.print("1.Encrypt\t2.Decrypt\t3.Exit\nEnter your choice: ");
            int choice = s.nextInt();
            switch(choice)
            {
                case 1:
                    encrypt(word);
                    break;

                case 2:
                    decrypt(word);
                    break;

                case 3:
                    System.exit(1);
            }
        }
    }
}
```

OUTPUT:

```
Enter a word: helloworld
1.Encrypt       2.Decrypt       3.Exit
Enter your choice: 1
Cipher Text: hdellrloow

Enter a word: hdellrloow
1.Encrypt       2.Decrypt       3.Exit
Enter your choice: 2
Plain text: helloworld
```

RESULT:

Thus, encryption and decryption using Transposition Cipher has been performed successfully and the output is verified.

EXP NO :2.a **DATE:** 18/08/2023	**CRYPTANALYSIS ON CAESER CIPHER**

AIM:

To perform cryptographic attack on the cipher-text generated using Caeser cipher.

CODE:

```
import java.util.Scanner;

public class caeser_attack {
   public static void main(String[] args) {
      String alphabets = "abcdefghijklmnopqrstuvwxyz";
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter cipher to be attacked:");
      String cipher = scanner.nextLine();
      for (int i = 0; i < 26; i++) {
         StringBuilder brute = new StringBuilder();
         for (char c : cipher.toCharArray()) {
            int num=alphabets.indexOf(c) - i;
            if(num<0)   num +=26;
            brute.append(alphabets.charAt((((num) % 26)) % 26));
         }
         System.out.println("Key used:" + i + " Plain:" + brute);
      }
   }
}
```

OUTPUT:

```
Enter cipher to be attacked:mjqqtbtwqi
Key used:0 Plain:mjqqtbtwqi
Key used:1 Plain:lippsasvph
Key used:2 Plain:khoorzruog
Key used:3 Plain:jgnnqyqtnf
Key used:4 Plain:ifmmpxpsme
Key used:5 Plain:helloworld
Key used:6 Plain:gdkknvnqkc
Key used:7 Plain:fcjjmumpjb
Key used:8 Plain:ebiiltloia
Key used:9 Plain:dahhksknhz
Key used:10 Plain:czggjrjmgy
```

RESULT:

Thus, cryptographic attack on the cipher-text generated has been performed successfully using Caeser cipher and the output is verified.

EXP NO :2.b **DATE:** 18/08/2023	**CRYPTANALYSIS ON AFFINE CIPHER**

AIM:

To perform cryptographic attack on the cipher-text generated using Affine cipher.

CODE:

```
import java.util.Scanner;

public class affine_attack {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String alphabetsLower = "abcdefghijklmnopqrstuvwxyz";
        String alphabetsUpper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        int aInverse, bInverse;

        System.out.print("Enter cipher to be attacked:");
        String cipher = scanner.nextLine();

        for (int a = 1; a < 26; a++) {
            aInverse = findInverse(a, 26);
            if (aInverse == -1) {
                continue; // Skip keys without inverses
            }

            for (int b = 0; b < 26; b++) {
                StringBuilder plain = new StringBuilder();

                for (char c : cipher.toCharArray()) {
                    if (Character.isLowerCase(c)) {
                        int index = alphabetsLower.indexOf(c);
                        if (index != -1) {
                            int newIndex = (aInverse * (index - b)) % 26;
                            if (newIndex < 0) {
                                newIndex += 26;
                            }
                            plain.append(alphabetsLower.charAt(newIndex));
                        } else {
                            plain.append(c);
                        }
                    } else if (Character.isUpperCase(c)) {
                        int index = alphabetsUpper.indexOf(c);
                        if (index != -1) {
                            int newIndex = (aInverse * (index - b)) % 26;
                            if (newIndex < 0) {
                                newIndex += 26;
```

```
                    }
                    plain.append(alphabetsUpper.charAt(newIndex));
                } else {
                    plain.append(c);
                }
            } else {
                plain.append(c);
            }
        }

        System.out.println("Key a: " + a + " Key b: " + b + " Plain: " + plain.toString());
    }
  }

  scanner.close();
 }

 public static int findInverse(int a, int m) {
  for (int i = 0; i < m; i++) {
   if ((a * i) % m == 1) {
    return i;
   }
  }
  return -1;
 }
}
```

OUTPUT:

```
Enter cipher to be attacked:jkzzyeyxzt
Key a: 1 Key b: 0 Plain: jkzzyeyxzt
Key a: 1 Key b: 1 Plain: ijyyxdxwys
Key a: 1 Key b: 2 Plain: hixxwcwvxr
Key a: 1 Key b: 3 Plain: ghwwvbvuwq
Key a: 1 Key b: 4 Plain: fgvvuautvp
Key a: 1 Key b: 5 Plain: efuutztsuo
Key a: 1 Key b: 6 Plain: dettsysrtn
Key a: 1 Key b: 7 Plain: cdssrxrqsm
Key a: 1 Key b: 8 Plain: bcrrqwqprl
Key a: 1 Key b: 9 Plain: abqqpvpoqk
Key a: 1 Key b: 10 Plain: zappouonpj
Key a: 1 Key b: 11 Plain: yzoontnmoi
Key a: 1 Key b: 12 Plain: xynnmsmlnh
Key a: 1 Key b: 13 Plain: wxmmlrlkmg
Key a: 1 Key b: 14 Plain: vwllkqkjlf
Key a: 1 Key b: 15 Plain: uvkkjpjike
```

```
Key a: 17 Key b: 14 Plain: pmttwewztl
Key a: 17 Key b: 15 Plain: spwwzhzcwo
Key a: 17 Key b: 16 Plain: vszzckcfzr
Key a: 17 Key b: 17 Plain: yvccfnficu
Key a: 17 Key b: 18 Plain: byffiqilfx
Key a: 17 Key b: 19 Plain: ebiiltloia
Key a: 17 Key b: 20 Plain: helloworld
Key a: 17 Key b: 21 Plain: khoorzruog
Key a: 17 Key b: 22 Plain: nkrrucuxrj
Key a: 17 Key b: 23 Plain: qnuuxfxaum
Key a: 17 Key b: 24 Plain: tqxxaiadxp
Key a: 17 Key b: 25 Plain: wtaadldgas
Key a: 19 Key b: 0 Plain: vgppesetpb
Key a: 19 Key b: 1 Plain: kveethtieq
Key a: 19 Key b: 2 Plain: zkttiwixtf
```

RESULT:

Thus, cryptographic attack on the cipher-text generated has been performed successfully using Affine cipher and the output is verified.

EXP NO :2.c DATE: 25/08/2023	CRYPTANALYSIS ON HILL CIPHER

AIM:

To perform cryptographic attack on the cipher-text generated using Hill cipher.

CODE:

```
import java.util.Scanner;

import Jama.Matrix;

public class hill_attack {
   public static int mul_inv(int a, int b) {
      int t1 = 0, t2 = 1, t;
      int r, q;
      while (b != 0) {
         q = a / b;
         r = a % b;
         a = b;
         b = r;
         t = t1 - q * t2;
         t1 = t2;
         t2 = t;

      }
      if (t1 < 0)
         return t1 + 26;
      return t1;
   }
   static int[][] multiplyMatrix(int row1, int col1,
         int A[][], int row2,
         int col2, int B[][]) {
      int i, j, k;
      int C[][] = new int[row1][col2];
      for (i = 0; i < row1; i++) {
         for (j = 0; j < col2; j++) {
            for (k = 0; k < row2; k++)
               C[i][j] += A[i][k] * B[k][j];
         }
      }
      return C;
   }
   public static int[][] matInverse(double[][] mat1){
      Matrix m1 = new Matrix(mat1);
      Matrix m1i = m1.inverse();
      double[][] inverseArray = m1i.getArray();
```

```
        int[][] ansArray=new int[inverseArray.length][inverseArray[0].length];
        for(int i=0;i<inverseArray.length;i++){
            for(int j=0;j<inverseArray[i].length;j++){
                int ans=(int) (Math.round(inverseArray[i][j] * m1.det()));
                ans%=26;
                if(ans<0) ans+=26;
                ans=ans*mul_inv(26,(int)(Math.round(m1.det())));
                ans%=26;
                if(ans<0) ans+=26;
                ansArray[i][j]=ans;
            }
        }
        return ansArray;
    }
    public static int GCD(int a, int b) {
        if (b == 0)
            return a;
        return (GCD(b, a % b));
    }
    public static void CryptAnalysis(String s, String t) {

        int mod=26;
        for (int a = 0; a < mod; a++) {
            for (int b = 0; b < mod; b++) {
                for (int c = 0; c < mod; c++) {
                    for (int d = 0; d < mod; d++) {
                        int determinant = (a * d - b * c) % mod;
                        if (determinant != 0 && GCD(determinant, mod) == 1) {
                            StringBuilder k=new StringBuilder();
                            k.append((char)('A' + a)).append((char) ('A' + b)).append((char) ('A' +
c)).append((char) ('A' + d));
                            String res = Decryption(s,k.toString());
                            System.out.println(res+" Key:"+k);
                            if(res.equals(t))
                            {
                                System.out.println("Key Found!");
                                System.exit(0);
                            }


                        }
                    }
                }
            }
        }
    }
```

```
    public static String Decryption(String s, String key) {
    int k = key.length();
    int n = s.length();
    int keySize = (int) Math.sqrt(k);
    int strSize;
    if (n % keySize == 0)
       strSize = n / keySize;
    else
       strSize = n / keySize + 1;
    double[][] keyMatrix = new double[(int) Math.sqrt(k)][(int) Math.sqrt(k)];
    int[][] cipherText = new int[keySize][strSize];

    int index = 0;
    for (int i = 0; i < keySize; i++) {
       for (int j = 0; j < keySize; j++) {
          int temp=key.charAt(index++) - 'A';
          keyMatrix[i][j] = temp;
       }
    }
    index = 0;
    for (int i = 0; i < strSize; i++) {
       for (int j = 0; j < keySize; j++) {
          if (index < n)
             cipherText[j][i] = s.charAt(index++) - 'A';
          else
             cipherText[j][i] = '-';
       }
    }
    int[][] inv=matInverse(keyMatrix);

    int[][] result = multiplyMatrix(keySize, keySize,inv,keySize, strSize, cipherText  );

    String ans = "";
   for (int i = 0; i < result[0].length; i++) {
       for (int j = 0; j < result.length; j++) {
          result[j][i] = (result[j][i] % 26) + 'A';
          ans += (char) result[j][i];
       }
    }
    return ans;
}

public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Enter plain text: ");
    String plain = s.nextLine();
    System.out.print("Enter cipher text: ");
    String cipher = s.nextLine();
```

```
        CryptAnalysis(cipher,plain);
    }
}
```

OUTPUT:

```
Enter plain text: TEXT
Enter cipher text: RGWL
WRXW Key:ABFA
DRDW Key:ABFB
KRJW Key:ABFC
RRPW Key:ABFD
YRVW Key:ABFE
FRBW Key:ABFF
MRHW Key:ABFG
TRNW Key:ABFH
ARTW Key:ABFI
HRZW Key:ABFJ
ORFW Key:ABFK
VRLW Key:ABFL
CRRW Key:ABFM
DUBP Key:DDAZ
OJVV Key:DDBC
UDJH Key:DDBE
QHRZ Key:DDBG
SFND Key:DDBI
WBFL Key:DDBK
IPHJ Key:DDBM
MLZR Key:DDBQ
YZBP Key:DDBS
CVTX Key:DDBU
ETPB Key:DDBW
AXXT Key:DDBY
LMLF Key:DDCD
TEXT Key:DDCF
Key Found!
```

RESULT:

Thus, cryptographic attack on the cipher-text generated has been performed successfully using Hill cipher and the output is verified.

EXP NO :3.a **DATE:** 08/09/2023	**SIMPLIFIED DATA ENCRYPTION STANDARD (SDES)**

AIM:

To demonstrate symmetric key encryption process using SDES.

CODE:

```
import java.util.Scanner;
public class sdes {
        // int key[] = {1, 0, 1, 0, 0, 0, 0, 0, 1, 0};
    static int key[] = new int[10];
        int P10[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6 };
        int P8[] = { 6, 3, 7, 4, 8, 5, 10, 9 };
        int key1[] = new int[8];
        int key2[] = new int[8];
        int[] IP = { 2, 6, 3, 1, 4, 8, 5, 7 };
        int[] EP = { 4, 1, 2, 3, 2, 3, 4, 1 };
        int[] P4 = { 2, 4, 3, 1 };
        int[] IP_inv = { 4, 1, 3, 5, 7, 2, 8, 6 };

        int[][] S0 = { { 1, 0, 3, 2 },
                       { 3, 2, 1, 0 },
                       { 0, 2, 1, 3 },
                       { 3, 1, 3, 2 } };
        int[][] S1 = { { 0, 1, 2, 3 },
                       { 2, 0, 1, 3 },
                       { 3, 0, 1, 0 },
                       { 2, 1, 0, 3 } };
        void key_generation()
        {
                int key_[] = new int[10];

                for (int i = 0; i < 10; i++) {
                        key_[i] = key[P10[i] - 1];
                }

                int Ls[] = new int[5];
                int Rs[] = new int[5];

                for (int i = 0; i < 5; i++) {
                        Ls[i] = key_[i];
                        Rs[i] = key_[i + 5];
                }
```

```
		int[] Ls_1 = shift(Ls, 1);
		int[] Rs_1 = shift(Rs, 1);

		for (int i = 0; i < 5; i++) {
			key_[i] = Ls_1[i];
			key_[i + 5] = Rs_1[i];
		}

		for (int i = 0; i < 8; i++) {
			key1[i] = key_[P8[i] - 1];
		}

		int[] Ls_2 = shift(Ls, 2);
		int[] Rs_2 = shift(Rs, 2);

		for (int i = 0; i < 5; i++) {
			key_[i] = Ls_2[i];
			key_[i + 5] = Rs_2[i];
		}

		for (int i = 0; i < 8; i++) {
			key2[i] = key_[P8[i] - 1];
		}

		System.out.print("Round 1 Key :");

		for (int i = 0; i < 8; i++)
			System.out.print(key1[i]);

		System.out.print("\nRound 2 Key :");

		for (int i = 0; i < 8; i++)
			System.out.print(key2[i]);
	}

	int[] shift(int[] ar, int n)
	{
		while (n > 0) {
			int temp = ar[0];
			for (int i = 0; i < ar.length - 1; i++) {
				ar[i] = ar[i + 1];
			}
			ar[ar.length - 1] = temp;
			n--;
		}
		return ar;
	}
```

```
int[] encryption(int[] plaintext)
{
	int[] arr = new int[8];

	for (int i = 0; i < 8; i++) {
		arr[i] = plaintext[IP[i] - 1];
	}
	int[] arr1 = function_(arr, key1);

	int[] after_swap = swap(arr1, arr1.length / 2);

	int[] arr2 = function_(after_swap, key2);

	int[] ciphertext = new int[8];

	for (int i = 0; i < 8; i++) {
		ciphertext[i] = arr2[IP_inv[i] - 1];
	}

	return ciphertext;
}


String binary_(int val)
{
	if (val == 0)
		return "00";
	else if (val == 1)
		return "01";
	else if (val == 2)
		return "10";
	else
		return "11";
}

int[] function_(int[] ar, int[] key_)
{

	int[] l = new int[4];
	int[] r = new int[4];

	for (int i = 0; i < 4; i++) {
		l[i] = ar[i];
		r[i] = ar[i + 4];
	}

	int[] ep = new int[8];
```

```
for (int i = 0; i < 8; i++) {
	ep[i] = r[EP[i] - 1];
}

for (int i = 0; i < 8; i++) {
	ar[i] = key_[i] ^ ep[i];
}

int[] l_1 = new int[4];
int[] r_1 = new int[4];

for (int i = 0; i < 4; i++) {
	l_1[i] = ar[i];
	r_1[i] = ar[i + 4];
}

int row, col, val;

row = Integer.parseInt("" + l_1[0] + l_1[3], 2);
col = Integer.parseInt("" + l_1[1] + l_1[2], 2);
val = S0[row][col];
String str_l = binary_(val);

row = Integer.parseInt("" + r_1[0] + r_1[3], 2);
col = Integer.parseInt("" + r_1[1] + r_1[2], 2);
val = S1[row][col];
String str_r = binary_(val);

int[] r_ = new int[4];
for (int i = 0; i < 2; i++) {
	char c1 = str_l.charAt(i);
	char c2 = str_r.charAt(i);
	r_[i] = Character.getNumericValue(c1);
	r_[i + 2] = Character.getNumericValue(c2);
}
int[] r_p4 = new int[4];
for (int i = 0; i < 4; i++) {
	r_p4[i] = r_[P4[i] - 1];
}

for (int i = 0; i < 4; i++) {
	l[i] = l[i] ^ r_p4[i];
}

int[] output = new int[8];
for (int i = 0; i < 4; i++) {
	output[i] = l[i];
	output[i + 4] = r[i];
```

```
		}
		return output;
	}
	int[] swap(int[] array, int n)
	{
		int[] l = new int[n];
		int[] r = new int[n];

		for (int i = 0; i < n; i++) {
			l[i] = array[i];
			r[i] = array[i + n];
		}

		int[] output = new int[2 * n];
		for (int i = 0; i < n; i++) {
			output[i] = r[i];
			output[i + n] = l[i];
		}

		return output;
	}

	int[] decryption(int[] ar)
	{
		int[] arr = new int[8];
		for (int i = 0; i < 8; i++)
		{
			arr[i] = ar[IP[i] - 1];
		}
		int[] arr1 = function_(arr, key2);
		int[] after_swap = swap(arr1, arr1.length / 2);
		int[] arr2 = function_(after_swap, key1);
		int[] decrypted = new int[8];
		for (int i = 0; i < 8; i++)
		{
			decrypted[i] = arr2[IP_inv[i] - 1];
		}

		return decrypted;
	}

	public static void main(String[] args)
	{
Scanner s = new Scanner(System.in);
		sdes obj = new sdes();
		int[] plaintext = new int[8];

System.out.print("\nEnter 8-bit plain text: ");
```

```
        String pt=s.nextLine();
        for(int i=0;i<8;i++)
        {
           plaintext[i] = pt.charAt(i)-48;
        }
        System.out.print("Enter 10-bit key: ");
        String k=s.nextLine();
        for(int i=0;i<10;i++)
        {
           key[i] = k.charAt(i)-48;
        }
                obj.key_generation();


                int[] ciphertext = obj.encryption(plaintext);

                System.out.println();
                System.out.print("Your cipher Text is :");
                for (int i = 0; i < 8; i++)
                        System.out.print(ciphertext[i]);

                int[] decrypted = obj.decryption(ciphertext);

                System.out.println();
                System.out.print("Your decrypted Text is :");
                for (int i = 0; i < 8; i++)
                        System.out.print(decrypted[i]);
        }
}
```

OUTPUT:

```
Enter 8-bit plain text: 10100011
Enter 10-bit key: 1001100110
Round 1 Key :10111000
Round 2 Key :10010011
Your cipher Text is :01011101
Your decrypted Text is :10100011
```

RESULT:

Thus, symmetric encryption and decryption using SDES has been performed successfully.

EXP NO :3.b **DATE:** 22/09/2023	**SIMPLIFIED ADVANCED ENCRYPTION STANDARD (SAES)**

AIM:

To demonstrate symmetric key encryption process using SAES.

CODE:

```
import java.util.Scanner;

public class saes {
   private static final int[] sBox = {0x9, 0x4, 0xA, 0xB, 0xD, 0x1, 0x8, 0x5, 0x6, 0x2, 0x0,
0x3, 0xC, 0xE, 0xF, 0x7};
   private static final int[] sBoxI = {0xA, 0x5, 0x9, 0xB, 0x1, 0x7, 0x8, 0xF, 0x6, 0x0, 0x2,
0x3, 0xC, 0x4, 0xD, 0xE};

   private int[] preRoundKey;
   private int[] round1Key;
   private int[] round2Key;

   public saes(int key) {
      int[][] roundKeys = keyExpansion(key);
      preRoundKey = roundKeys[0];
      round1Key = roundKeys[1];
      round2Key = roundKeys[2];
   }

   private int subWord(int word) {
      return (sBox[(word >> 4)] << 4) + sBox[word & 0x0F];
   }

   private int rotWord(int word) {
      return ((word & 0x0F) << 4) + ((word & 0xF0) >> 4);
   }

   private int[][] keyExpansion(int key) {
      int Rcon1 = 0x80;
      int Rcon2 = 0x30;

      int[] w = new int[6];
      w[0] = (key & 0xFF00) >> 8;
      w[1] = key & 0x00FF;
      w[2] = w[0] ^ (subWord(rotWord(w[1])) ^ Rcon1);
      w[3] = w[2] ^ w[1];
      w[4] = w[2] ^ (subWord(rotWord(w[3])) ^ Rcon2);
      w[5] = w[4] ^ w[3];
```

```
    return new int[][] {
        int_to_state((w[0] << 8) + w[1]),
        int_to_state((w[2] << 8) + w[3]),
        int_to_state((w[4] << 8) + w[5])
    };
}

private int gf_mult(int a, int b) {
    int product = 0;
    a = a & 0x0F;
    b = b & 0x0F;

    while (a != 0 && b != 0) {
        if ((b & 1) != 0) {
            product ^= a;
        }
        a <<= 1;
        if ((a & (1 << 4)) != 0) {
            a ^= 0b10011;
        }
        b >>= 1;
    }

    return product;
}

private int[] int_to_state(int n) {
    return new int[] {(n >> 12) & 0xF, (n >> 4) & 0xF, (n >> 8) & 0xF, n & 0xF};
}

private int state_to_int(int[] state) {
    return (state[0] << 12) + (state[2] << 8) + (state[1] << 4) + state[3];
}

private int[] add_round_key(int[] s1, int[] s2) {
    int[] result = new int[s1.length];
    for (int i = 0; i < s1.length; i++) {
        result[i] = s1[i] ^ s2[i];
    }
    return result;
}

private int[] sub_nibbles(int[] sbox, int[] state) {
    int[] result = new int[state.length];
    for (int i = 0; i < state.length; i++) {
        result[i] = sbox[state[i]];
    }
```

```
        return result;
    }

    private int[] shift_rows(int[] state) {
        return new int[]{state[0], state[1], state[3], state[2]};
    }

    private int[] mix_columns(int[] state) {
        return new int[]{
            state[0] ^ gf_mult(4, state[2]),
            state[1] ^ gf_mult(4, state[3]),
            state[2] ^ gf_mult(4, state[0]),
            state[3] ^ gf_mult(4, state[1])
        };
    }

    private int[] inverse_mix_columns(int[] state) {
        return new int[]{
            gf_mult(9, state[0]) ^ gf_mult(2, state[2]),
            gf_mult(9, state[1]) ^ gf_mult(2, state[3]),
            gf_mult(9, state[2]) ^ gf_mult(2, state[0]),
            gf_mult(9, state[3]) ^ gf_mult(2, state[1])
        };
    }

    public int encrypt(int plaintext) {
        int[] state = add_round_key(preRoundKey, int_to_state(plaintext));
        state = mix_columns(shift_rows(sub_nibbles(sBox, state)));
        state = add_round_key(round1Key, state);
        state = shift_rows(sub_nibbles(sBox, state));
        state = add_round_key(round2Key, state);
        return state_to_int(state);
    }

    public int decrypt(int ciphertext) {
        int[] state = add_round_key(round2Key, int_to_state(ciphertext));
        state = sub_nibbles(sBoxI, shift_rows(state));
        state = inverse_mix_columns(add_round_key(round1Key, state));
        state = sub_nibbles(sBoxI, shift_rows(state));
        state = add_round_key(preRoundKey, state);
        return state_to_int(state);
    }



    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
// int key = 0b0100101011110101;
```

```
        System.out.print("Enter key as binary: ");
        String str = s.nextLine();
        int key = Integer.parseInt(str,2);
        saes aes = new saes(key);

        System.out.print("Enter plain text in binary: ");
        str = s.nextLine();
        int plaintext = Integer.parseInt(str,2);

        int ciphertext = aes.encrypt(plaintext);
        int decryptedPlaintext = aes.decrypt(ciphertext);

        System.out.println("\n\nOriginal Plaintext: " + Integer.toBinaryString(plaintext));
        System.out.println("Cipher text: " + Integer.toBinaryString(ciphertext));
        System.out.println("Plain text (decrypted): " +
Integer.toBinaryString(decryptedPlaintext));
        s.close();
    }
}
```

OUTPUT:

```
Enter key as binary: 0100101011110101
Enter plain text in binary: 1101011100101000


Original Plaintext: 1101011100101000
Cipher text: 10010011101100
Plain text (decrypted): 1101011100101000
```

RESULT:

Thus, symmetric encryption and decryption using SAES has been performed successfully.

EXP NO : 4 **DATE:** 29/09/2023	**RSA ALGORITHM**

AIM:

To implement RSA algorithm and demonstrate the key generation and encryption process.

CODE:

```
import java.util.ArrayList;
import java.util.Scanner;
import java.math.BigInteger;
public class rsa {
   public static ArrayList<Integer> dec_to_bin_rev(int num)
   {
      ArrayList<Integer> res=new ArrayList<>();
      int x=num;
      while(x>0)
      {
         int rem = x%2;
         res.add(rem);
         x/=2;
      }
      return res;
   }

   public static int square_and_multiply(int a1, int x, BigInteger n)
   {
      BigInteger y = BigInteger.ONE;
      BigInteger a = BigInteger.valueOf(a1);
      ArrayList<Integer> binary=dec_to_bin_rev(x);
      // System.out.println(dec_to_bin_rev(b));
      for(int i=0;i<binary.size();i++)
      {
         if(binary.get(i)==1)
         {
            y = a.multiply(y).mod(n);
            a = a.multiply(a).mod(n);
         }
         else if(binary.get(i)==0)
         {
            a = a.multiply(a).mod(n);

         }
         // System.out.println(y + "\t" + a);
```

```
    }
    return y.intValue();
}


public static int extended_eucledian(int e, BigInteger n)
{
    BigInteger r1 = n;
    BigInteger r2 = BigInteger.valueOf(e);
    // System.out.println(r1+"\t"+r2);
    BigInteger r = r1.mod(r2);
    BigInteger q = r1.divide(r2);
    BigInteger t1 = BigInteger.valueOf(0);
    BigInteger t2 = BigInteger.valueOf(1);
    BigInteger t = t1.subtract(q.multiply(t2));
    // System.out.println(q+"\t"+r1+"\t"+r2+"\t"+r+"\t"+t1+"\t"+t2+"\t"+t);
    // while(r!=0)
    while(!r.equals(BigInteger.ZERO))
    {
        r1=r2;
        r2=r;
        t1=t2;
        t2=t;
        q=r1.divide(r2);
        r=r1.mod(r2);
        t=t1.subtract(q.multiply(t2));
        // System.out.println(q+"\t"+r1+"\t"+r2+"\t"+r+"\t"+t1+"\t"+t2+"\t"+t);

    }
    if(t2.compareTo(BigInteger.ZERO) == -1)    return t2.add(n).intValue();
    else return t2.intValue();
}

public static boolean check_prime(int n)
{
    if(n==0 || n==1 || n==2)    return false;
    for(int i=2;i<n;i++)
    {
        if(n%i==0)  return false;
    }
    return true;
}

public static ArrayList<Integer> encrypt(String plain_text, int e, BigInteger n)
{
    plain_text = plain_text.toLowerCase();
    char[] plain_char = plain_text.toCharArray();
    ArrayList<Integer> cipher_text = new ArrayList<>();
```

```
        // for(int i=0;i<plain_text.length();i++)
        for(char c: plain_char)
        {
            cipher_text.add(square_and_multiply(c-96, e, n));
        }
        return cipher_text;
    }

    public static ArrayList<Character> decrypt(ArrayList<Integer> cipher_text, int e,
BigInteger n, int p, int q)
    {
        int d = extended_eucledian(e,BigInteger.valueOf((p-1)*(q-1)));
        System.out.println("The private key (d) is "+d);
        ArrayList<Character> decrypted_text = new ArrayList<>();
        for(int i: cipher_text)
        {
            int c = square_and_multiply(i, d, n) + 96;
            decrypted_text.add((char)c);
        }
        return decrypted_text;
    }


    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);

        //Plain text
        System.out.print("Enter plain text: ");
        String plain_text = s.nextLine();

        //Value p
        System.out.print("Enter value of p: ");
        int p = Integer.parseInt(s.nextLine());
        if(!check_prime(p))
        {
            System.out.println("Enter a prime number");
            System.exit(0);
        }

        //Value q
        System.out.print("Enter value of q: ");
        int q = Integer.parseInt(s.nextLine());
        if(!check_prime(q))
        {
            System.out.println("Enter a prime number");
            System.exit(0);
        }
```

```
        BigInteger n = BigInteger.valueOf(p*q);

        //Public key e
        System.out.print("Enter public key (e): ");
        int e = Integer.parseInt(s.nextLine());
        if(e<=0 || e>=(p-1)*(q-1))
        {
           System.out.print("Enter correct value for e (0 to phi(n))");
           System.exit(0);
        }

        //Encryption
        ArrayList<Integer> cipher_text = encrypt(plain_text,e,n);
        System.out.println("\nThe cipher text is "+cipher_text);

        //Decryption
        ArrayList<Character> decrypted_text = decrypt(cipher_text, e, n, p, q);
        System.out.print("The decrypted text is ");
        for(char c: decrypted_text) System.out.print(c);

        s.close();
    }
}
```

OUTPUT:

```
Enter plain text: abcdef
Enter value of p: 1009
Enter value of q: 1031
Enter public key (e): 13

The cipher text is [1, 8192, 554044, 531008, 455858, 1031450]
The private key (d) is 638917
The decrypted text is abcdef
```

RESULT:

Thus, RSA algorithm has been implemented and verified successfully.

EXP NO : 5 **DATE:** 06/10/2023	**SHA ALGORITHM**

AIM:

To generate message digest for the given message using the SHA algorithm and verify the integrity of message.

CODE:

```
import hashlib
import binascii
import struct

initial_hash = (
   0x6a09e667f3bcc908,   0xbb67ae8584caa73b,   0x3c6ef372fe94f82b,
   0xa54ff53a5f1d36f1,   0x510e527fade682d1,   0x9b05688c2b3e6c1f,
   0x1f83d9abfb41bd6b,   0x5be0cd19137e2179,
)
round_constants = (
   0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f,
   0xe9b5dba58189dbbc, 0x3956c25bf348b538, 0x59f111f1b605d019,
   0x923f82a4af194f9b, 0xab1c5ed5da6d8118, 0xd807aa98a3030242,
   0x12835b0145706fbe, 0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2,
   0x72be5d74f27b896f, 0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
   0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3,
   0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65, 0x2de92c6f592b0275,
   0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4, 0x76f988da831153b5,
   0x983e5152ee66dfab, 0xa831c66d2db43210, 0xb00327c898fb213f,
   0xbf597fc7beef0ee4, 0xc6e00bf33da88fc2, 0xd5a79147930aa725,
   0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc,
   0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed, 0x53380d139d95b3df,
   0x650a73548baf63de, 0x766a0abb3c77b2a8, 0x81c2c92e47edaee6,
   0x92722c851482353b, 0xa2bfe8a14cf10364, 0xa81a664bbc423001,
   0xc24b8b70d0f89791, 0xc76c51a30654be30, 0xd192e819d6ef5218,
   0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8,
   0x19a4c116b8d2d0c8, 0x1e376c085141ab53, 0x2748774cdf8eeb99,
   0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63, 0x4ed8aa4ae3418acb,
   0x5b9cca4f7763e373, 0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc,
   0x78a5636f43172f60, 0x84c87814a1f0ab72, 0x8cc702081a6439ec,
   0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915,
   0xc67178f2e372532b, 0xca273eceea26619c, 0xd186b8c721c0c207,
   0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178, 0x06f067aa72176fba,
   0x0a637dc5a2c898a6, 0x113f9804bef90dae, 0x1b710b35131c471b,
   0x28db77f523047d84, 0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
   0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a,
```

```
    0x5fcb6fab3ad6faec, 0x6c44198c4a475817,
)
def _right_rotate(n: int, bits: int) -> int:
    return (n >> bits) | (n << (64 - bits)) & 0xFFFFFFFFFFFFFFFF

def sha_512(message: str) -> str:
    if type(message) is not str:
        raise TypeError('Given message should be a string.')
    message_array = bytearray(message, encoding='utf-8')

    mdi = len(message_array) % 128
    padding_len = 119 - mdi if mdi < 112 else 247 - mdi
    ending = struct.pack('!Q', len(message_array) << 3)
    message_array.append(0x80)
    message_array.extend([0] * padding_len)
    message_array.extend(bytearray(ending))

    sha512_hash = list(initial_hash)
    for chunk_start in range(0, len(message_array), 128):
        chunk = message_array[chunk_start:chunk_start + 128]

        w = [0] * 80
        w[0:16] = struct.unpack('!16Q', chunk)

        for i in range(16, 80):
            s0 = (
                _right_rotate(w[i - 15], 1) ^
                _right_rotate(w[i - 15], 8) ^
                (w[i - 15] >> 7)
            )
            s1 = (
                _right_rotate(w[i - 2], 19) ^
                _right_rotate(w[i - 2], 61) ^
                (w[i - 2] >> 6)
            )
            w[i] = (w[i - 16] + s0 + w[i - 7] + s1) & 0xFFFFFFFFFFFFFFFF

        a, b, c, d, e, f, g, h = sha512_hash

        for i in range(80):
            sum1 = (
                _right_rotate(e, 14) ^
                _right_rotate(e, 18) ^
                _right_rotate(e, 41)
            )
            ch = (e & f) ^ (~e & g)
            temp1 = h + sum1 + ch + round_constants[i] + w[i]
            sum0 = (
```

```
                _right_rotate(a, 28) ^
                _right_rotate(a, 34) ^
                _right_rotate(a, 39)
            )
            maj = (a & b) ^ (a & c) ^ (b & c)
            temp2 = sum0 + maj

            h = g
            g = f
            f = e
            e = (d + temp1) & 0xFFFFFFFFFFFFFFFF
            d = c
            c = b
            b = a
            a = (temp1 + temp2) & 0xFFFFFFFFFFFFFFFF

        sha512_hash = [
            (x + y) & 0xFFFFFFFFFFFFFFFF
            for x, y in zip(sha512_hash, (a, b, c, d, e, f, g, h))
        ]

    return binascii.hexlify(
        b"".join(struct.pack('!Q', element) for element in sha512_hash),
    ).decode('utf-8')

if __name__ == "__main__":
    user_input = input("Enter a string: ")
    custom_hash = sha_512(user_input)
    hashlib_hash = hashlib.sha512(user_input.encode()).hexdigest()
    if custom_hash == hashlib_hash:
        print("SHA 512 Encryption completed successfully\nEncrypted hash:")
        print(custom_hash)
    else:
        print("Custom hash and hashlib hash do not match")
```

OUTPUT:

```
Enter a string: hello, world
SHA 512 Encryption completed successfully
Encrypted hash:
8710339dcb6814d0d9d2290ef422285c9322b7163951f9a0ca8f883d3305286f44139aa374848e4174f5aada663027e4548637b6d19894aec4fb6c46a139fbf9
```

RESULT:

Thus, generation of message digest for the given message using the SHA algorithm has been performed successfully and the output is verified