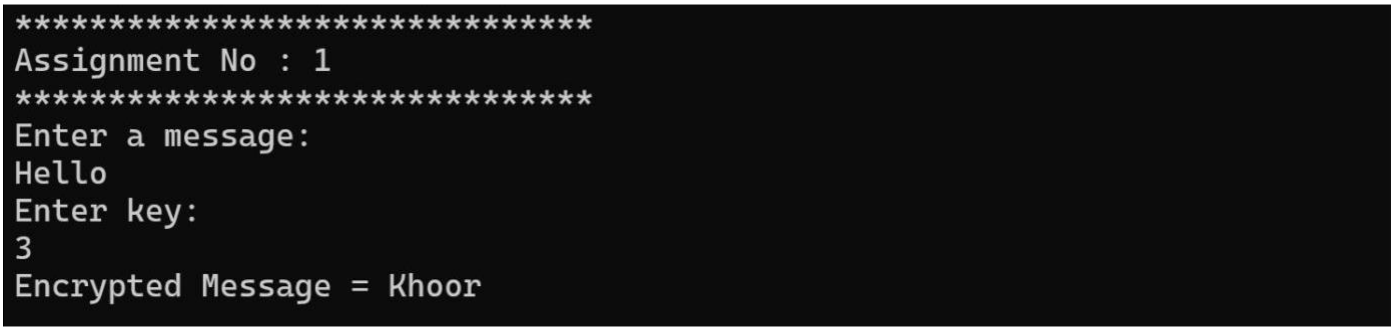
Assignment 1

```
import java.util.Scanner;

public class Assignment1

{

    public static void main(String s[])

    {

        String message, encryptedMessage = "";

        int key;

        char ch;

        Scanner sc = new Scanner(System.in);

        System.out.println("*******************************");

        System.out.println("Assignment No : 1");

        System.out.println("*******************************");

        System.out.println("Enter a message: "); message =

        sc.nextLine();

        System.out.println("Enter key: ");

        key = sc.nextInt();

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

        {

            ch = message.charAt(i);

            if(ch >= 'a' && ch <='z')

            {

                ch = (char)(ch + key);

                if(ch > 'z')

                {

                    ch = (char)(ch - 'z' + 'a' - 1);

                }

                encryptedMessage += ch;

            }

            else

            if(ch >= 'A' && ch <= 'Z')

            {
                ch = (char)(ch + key);
if(ch > 'Z')
                {
                    ch = (char)(ch - 'Z' + 'A' - 1);
                }
                encryptedMessage += ch;
            }
            else
            {
                encryptedMessage += ch;
            }
        }
        System.out.println("Encrypted Message = " + encryptedMessage);
    }
}
```



```
*******************************
Assignment No : 1
*******************************
Enter a message:
Hello
Enter key:
3
Encrypted Message = Khoor
```

Assignment 2

```
package course;

import java.util.Arrays;

import java.util.Scanner;

public class Assignment2

  {

      private static char[][] keySquare;

      private static void
generateKeySquare(String key)

      {

         key = key.replace("J",
"I").toUpperCase();

         key = key.replaceAll("[^A-Z]", "");

         String alphabet =
"ABCDEFGHIKLMNOPQRSTUVWXYZ"; String

         combinedKey = key + alphabet;

         combinedKey =
combinedKey.replaceAll("(.)(?=.*\\1)", ""); //
Remove duplicate characters

         keySquare = new

         char[5][5]; int rowIndex =

         0; int colIndex = 0;

         for (char ch :
combinedKey.toCharArray())

         {

            keySquare[rowIndex][colIndex] =

            ch; colIndex++;


            if (colIndex == 5)

            {

               colIndex = 0;

               rowIndex++;

            }
```

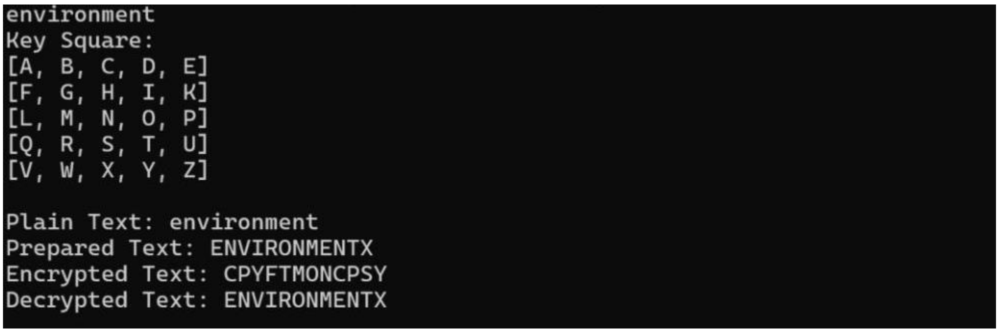
```
      private static String
preparePlainText(String plainText)

      {

         plainText = plainText.replace("J",
"I").toUpperCase();

         plainText =
plainText.replaceAll("[^A-Z]", "");

         StringBuilder preparedText =
new StringBuilder(plainText);


         for (int i = 0; i < preparedText.length(); i
+= 2)

         {

            if (i + 1 == preparedText.length())


            {

               preparedText.append('X');

            }

            else if (preparedText.charAt(i)
== preparedText.charAt(i + 1))

            {

               preparedText.insert(i + 1, 'X');

            }

         }

         return preparedText.toString();

      }

      private static String
encrypt(String plainText)

      {

         StringBuilder encryptedText =
new StringBuilder();
```

```
        for (int i = 0; i < plainText.length(); i +=
2)

        {

          char ch1 = plainText.charAt(i);

          char ch2 = plainText.charAt(i + 1);

          int row1 = -1, col1 = -1, row2 = -1,
col2 = -1

          char encryptedCh1, encryptedCh2;

          if (row1 == row2)

          {

            encryptedCh1 =
keySquare[row1][(col1 + 1) % 5];

            encryptedCh2 =
keySquare[row2][(col2 + 1) % 5];

          }

          else if (col1 == col2)

          {

            encryptedCh1 = keySquare[(row1
+ 1) % 5][col1];

            encryptedCh2 = keySquare[(row2 +
1) % 5][col2];

          }

          else

          {

            encryptedCh1 =
keySquare[row1][col2];

            encryptedCh2 =
keySquare[row2][col1];

          }



encryptedText.append(encryptedCh1).append
(encryptedCh2);

        }


        return encryptedText.toString();

      }
```

```
      private static String
decrypt(String encryptedText)

      {

        StringBuilder decryptedText =
new StringBuilder();


        for (int i = 0; i < encryptedText.length();
i += 2)

        {

          char ch1 = encryptedText.charAt(i);

          char ch2 = encryptedText.charAt(i +
1);

          int row1 = -1, col1 = -1, row2 = -
1, col2 = -1;


          for (int row = 0; row < 5; row++)

          {

            for (int col = 0; col < 5; col++)

            {

              if (keySquare[row][col] == ch1)

              {

                row1 = row;

                col1 = col;

              }


              if (keySquare[row][col] == ch2)

              {

                row2 = row;

                col2 = col;

              }

            }

          }


            decryptedCh2 = keySquare[(row2 +
4) % 5][col2];

          }

          else
```

```
            {
                decryptedCh1 =
keySquare[row1][col2];
                decryptedCh2 =
keySquare[row2][col1];
            }



decryptedText.append(decryptedCh1).append
(decryptedCh2);
        }


        return decryptedText.toString();
    }


public static void main(String[] args)
{
   String key = "KEYWORD";
   generateKeySquare(key);
   Scanner scan = new Scanner(System.in);
// Take input from user using scanner class
   String plainText = scan.nextLine();
   String preparedText =
preparePlainText(plainText);
   String encryptedText =
encrypt(preparedText);
```

```
   String decryptedText =
decrypt(encryptedText);
   System.out.println("Key Square:");


   for (char[] row : keySquare)
   {
      System.out.println(Arrays.toString(row));
   }


   System.out.println("\nPlain Text: "
+ plainText);
   System.out.println("Prepared Text: "
+ preparedText);
   System.out.println("Encrypted Text: "
+ encryptedText);
   System.out.println("Decrypted Text: "
+ decryptedText);
}
}
```



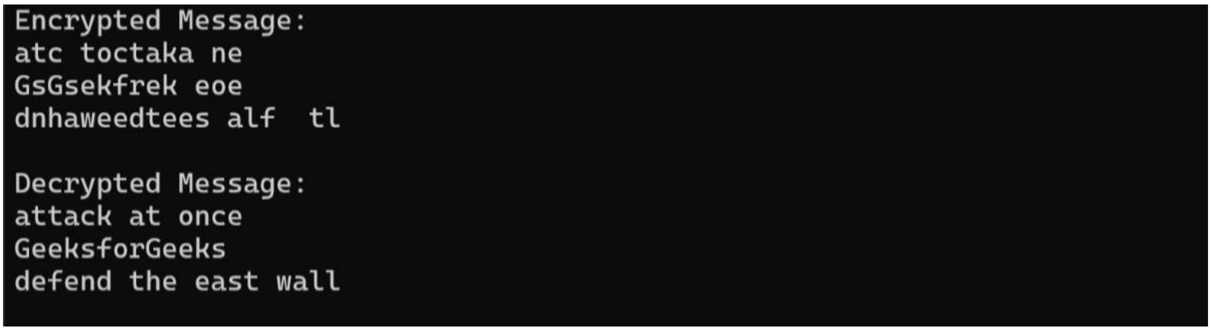
```
environment
Key Square:
[A, B, C, D, E]
[F, G, H, I, K]
[L, M, N, O, P]
[Q, R, S, T, U]
[V, W, X, Y, Z]

Plain Text: environment
Prepared Text: ENVIRONMENTX
Encrypted Text: CPYFTMONCPSY
Decrypted Text: ENVIRONMENTX
```

Assignment 3

```
import java.util.Arrays;

class Assignment3

{

   // function to encrypt a message

   public static String encryptRailFence(String text, int key)

   {

      // create the matrix to cipher plain text

      // key = rows , length(text) = columns

      char[][] rail = new char[key][text.length()];

      // filling the rail matrix to distinguish filled

      // spaces from blank ones

      for (int i = 0; i < key; i++)

         Arrays.fill(rail[i], '\n');

      boolean dirDown = false;

      int row = 0, col = 0;

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

         // check the direction of flow

         // reverse the direction if we've just

         // filled the top or bottom rail

         if (row == 0 || row == key - 1)

            dirDown = !dirDown;

         // fill the corresponding alphabet

         rail[row][col++] = text.charAt(i);

         // find the next row using direction

         flag if (dirDown)

            row++;

         else

            row--;

      }

      // now we can construct the cipher using the rail

      // matrix

      StringBuilder result = new StringBuilder();

      for (int i = 0; i < key; i++)

         for (int j = 0; j < text.length(); j++)
```

```
            if (rail[i][j] != '\n')
                result.append(rail[i][j]);
    return result.toString();
}
// This function receives cipher-text and key
// and returns the original text after decryption
public static String decryptRailFence(String cipher, int key)
{
        Arrays.fill(rail[i], '\n');
    // to find the direction
    boolean dirDown = true;
    int row = 0, col = 0;
    // mark the places with '*'
    for (int i = 0; i < cipher.length(); i++) {
        // check the direction of
        flow if (row == 0)
            dirDown = true;
        if (row == key - 1)
        dirDown = false;
        // place the marker
        rail[row][col++] = '*';
        // find the next row using direction
        flag if (dirDown)
            row++;
        else
            row--;
    }
    // now we can construct the fill the rail
    matrix int index = 0;
    for (int i = 0; i < key; i++)
        for (int j = 0; j < cipher.length();
            j++) if (rail[i][j] == '*'
                && index < cipher.length())
                rail[i][j] = cipher.charAt(index++);
    StringBuilder result = new StringBuilder();
```

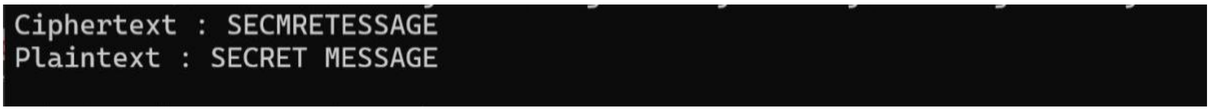
```
        row = 0;

        col = 0;

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

            // check the direction of

            flow if (row == 0)

                dirDown = true;

            if (row == key - 1)

            dirDown = false;

            if (rail[row][col] != '*')

                result.append(rail[row][col++]);

            // find the next row using direction

    flag public static void main(String[] args)

    {

        // Encryption System.out.println("Encrypted Message: ");

        System.out.println(encryptRailFence("attack at once", 2));

        System.out.println( encryptRailFence("GeeksforGeeks ", 3));

        System.out.println(encryptRailFence("defend the east wall",

        3));

        // Now decryption of the same cipher-text

        System.out.println("\nDecrypted Message: ");

        System.out.println(decryptRailFence("atc toctaka ne", 2));

        System.out.println(decryptRailFence("GsGsekfrek eoe", 3));

        System.out.println(decryptRailFence("dnhaweedtees alf tl", 3));

    }

}
```



```
Encrypted Message:
atc toctaka ne
GsGsekfrek eoe
dnhaweedtees alf  tl

Decrypted Message:
attack at once
GeeksforGeeks
defend the east wall
```

Assignment 4

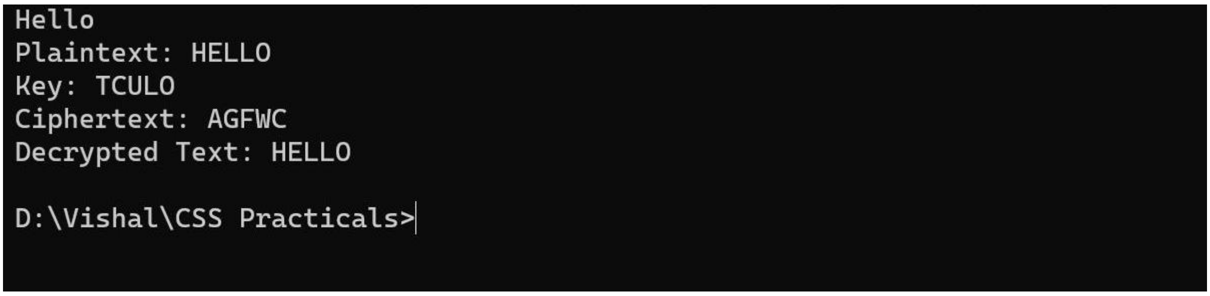
```
public class Assignment4 {
    public static String encrypt(String message, String keyword) {
        // Create a matrix to store the plaintext
        message. int keyLength = keyword.length();
        char[][] matrix = new char[keyLength][message.length()];

        // Write the plaintext message to the matrix.
        int row = 0;
        int col = 0;
        for (int i = 0; i < message.length(); i++) {
            matrix[row][col] = message.charAt(i);
            row++;
            if (row == keyLength) {
                row = 0;
                col++;
            }
        }

        // Order the columns by the alphabetical order of the
        keyword. int[] columnOrder = new int[keyLength];
        for (int i = 0; i < keyLength; i++)
            { columnOrder[i] = i;
        }

        // Sort the column order.
        Arrays.sort(columnOrder, (o1, o2) ->
Character.compare(keyword.charAt(o1), keyword.charAt(o2)));

        // Read the ciphertext off column by column, in the order specified by the column
        order. String ciphertext = "";
        for (int i = 0; i < keyLength; i++) {
            for (int j = 0; j < message.length(); j++) {
                ciphertext += matrix[j][columnOrder[i]];
            }
        }

        return ciphertext;
    }

    public static String decrypt(String ciphertext, String keyword) {
        // Create a matrix to store the ciphertext.
        int keyLength = keyword.length();
        char[][] matrix = new char[keyLength][ciphertext.length()];

        // Order the columns by the alphabetical order of the
        keyword. int[] columnOrder = new int[keyLength];
        for (int i = 0; i < keyLength; i++)
            { columnOrder[i] = i;
        }

        // Sort the column order.
        Arrays.sort(columnOrder, (o1, o2) ->
Character.compare(keyword.charAt(o1), keyword.charAt(o2)));

        // Write the ciphertext to the matrix, in the order specified by the column
        order. int row = 0;
        int col = 0;
        for (int i = 0; i < ciphertext.length(); i++) {
            matrix[row][columnOrder[col]] =
            ciphertext.charAt(i); col++;
            if (col == keyLength)
                { col = 0;
                row++;
            }
        }


        // Read the plaintext off row by row, from left to
        right. String plaintext = "";

        for (int i = 0; i < matrix[0].length; i++)
            { for (int j = 0; j < keyLength; j++) {
                plaintext += matrix[j][i];
            }
        }


        return plaintext;
    }


    public static void main(String[] args) {
        String message = "SECRET MESSAGE";
        String keyword = "ZEBRAS";


        String ciphertext = encrypt(message, keyword);
        System.out.println("Ciphertext: " + ciphertext);


        String plaintext = decrypt(ciphertext, keyword);
        System.out.println("Plaintext: " + plaintext);
    }
}
```



```
Ciphertext : SECMRETESSAGE
Plaintext : SECRET MESSAGE
```

Assignment 5

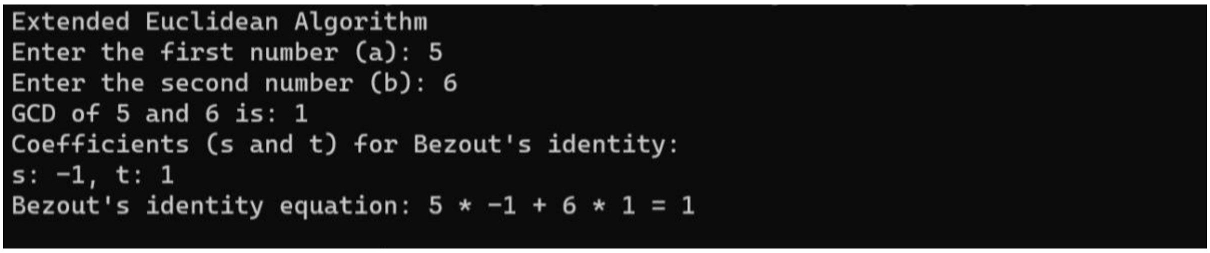
```
import java.util.Random;
import java.util.Scanner;
public class Assignment5 {
// Function to generate a random key (pad) of the same length as the
plaintext public static String generateRandomKey(int length) {
Random random = new Random();
StringBuilder keyBuilder = new
StringBuilder(); for (int i = 0; i < length; i++) {
char randomChar = (char) (random.nextInt(26) + 'A'); // Generates a random uppercase
letter keyBuilder.append(randomChar);
}
return keyBuilder.toString();
}
// Function to perform one-time pad encryption
public static String encrypt(String plaintext, String key)
{ if (plaintext.length() != key.length()) {
throw new IllegalArgumentException("Plaintext and key must have the same length.");
}
StringBuilder ciphertextBuilder = new StringBuilder();
for (int i = 0; i < plaintext.length(); i++) {
char encryptedChar = (char) ((plaintext.charAt(i) + key.charAt(i)) % 26 +
'A'); ciphertextBuilder.append(encryptedChar); }

return ciphertextBuilder.toString();
}
// Function to perform one-time pad decryption
public static String decrypt(String ciphertext, String key)
{ if (ciphertext.length() != key.length()) {
throw new IllegalArgumentException("Ciphertext and key must have the same length.");
}
StringBuilder decryptedBuilder = new StringBuilder();
for (int i = 0; i < ciphertext.length(); i++) {

char decryptedChar = (char) ((ciphertext.charAt(i) - key.charAt(i) + 26) % 26 +

'A'); decryptedBuilder.append(decryptedChar); }


return decryptedBuilder.toString();

}

public static void main(String[] args) {

// Input string from user

Scanner scan = new Scanner(System.in);

String randomtext = scan.nextLine();

String plaintext = randomtext.toUpperCase();

String key = generateRandomKey(plaintext.length());

System.out.println("Plaintext: " + plaintext);

System.out.println("Key: " + key);

String ciphertext = encrypt(plaintext, key);

System.out.println("Ciphertext: " + ciphertext);

String decryptedText = decrypt(ciphertext, key);

System.out.println("Decrypted Text: " + decryptedText);

}

}
```



```
Hello
Plaintext: HELLO
Key: TCULO
Ciphertext: AGFWC
Decrypted Text: HELLO

D:\Vishal\CSS Practicals>
```

Assignment 6

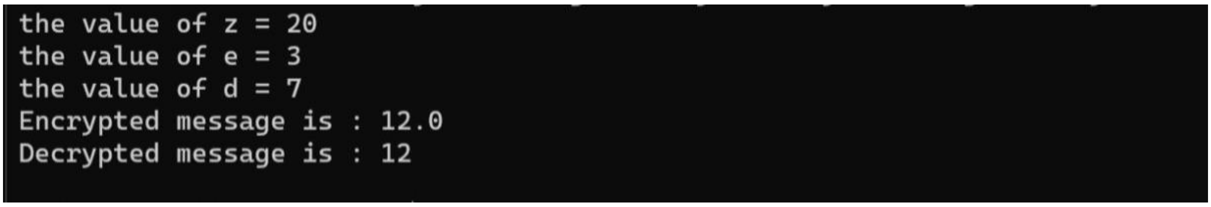
```
import java.util.Scanner;


public class Assignment6 {
// Function to perform the extended Euclidean algorithm
public static int[] extendedEuclidean(int a, int b) {
if (b == 0) {
return new int[]{a, 1, 0};
}
int[] values = extendedEuclidean(b, a %
b); int gcd = values[0];
int s = values[2];
int t = values[1] - (a / b) * values[2];
return new int[]{gcd, s, t};
}
public static void main(String[] args) { Scanner scanner
= new Scanner(System.in);
System.out.println("Extended Euclidean Algorithm");
System.out.print("Enter the first number (a): ");
int a = scanner.nextInt();
System.out.print("Enter the second number (b):
"); int b = scanner.nextInt();
scanner.close();
int[] values = extendedEuclidean(a,
b); int gcd = values[0];
int s = values[1];
int t = values[2];

System.out.println("GCD of " + a + " and " + b + " is: " + gcd);

System.out.println("Coefficients (s and t) for Bezout's identity:");

System.out.println("s: " + s + ", t: " + t);

System.out.println("Bezout's identity equation: " + a + " * " + s + " + " + b + " * " + t + " = " + gcd);

}

}
```



```
Extended Euclidean Algorithm
Enter the first number (a): 5
Enter the second number (b): 6
GCD of 5 and 6 is: 1
Coefficients (s and t) for Bezout's identity:
s: -1, t: 1
Bezout's identity equation: 5 * -1 + 6 * 1 = 1
```

Assignment 7

```
// Java Program to Implement the RSA
Algorithm import java.math.*;
import java.util.*;
class Assignment7{
    public static void main(String args[])
    {
        int p, q, n, z, d = 0, e, i;
        // The number to be encrypted and
        decrypted int msg = 12;
        double c; BigInteger
        msgback;
        // 1st prime number p
        p = 3;


        // 2nd prime number
        q q = 11;
        n = p * q;
        z = (p - 1) * (q - 1);
        System.out.println("the value of z = " +
        z); for (e = 2; e < z; e++) {


            // e is for public key
            exponent if (gcd(e, z) == 1) {
                break;
            }
        }
        System.out.println("the value of e = " + e);
        for (i = 0; i <= 9; i++) {
            int x = 1 + (i * z);


            // d is for private key exponent
            if (x % e == 0) {
                d = x / e;
                break;
            }
        }
        System.out.println("the value of d = " + d);
        c = (Math.pow(msg, e)) % n;
        System.out.println("Encrypted message is : " + c);
        // converting int value of n to BigInteger
        BigInteger N = BigInteger.valueOf(n);
        // converting float value of c to BigInteger BigInteger
        C = BigDecimal.valueOf(c).toBigInteger(); msgback =
        (C.pow(d)).mod(N); System.out.println("Decrypted
        message is : "
                                + msgback);
    }
    static int gcd(int e, int z)
    {
        if (e == 0)
            return z;
        else
            return gcd(z % e, e);
    }
}
```

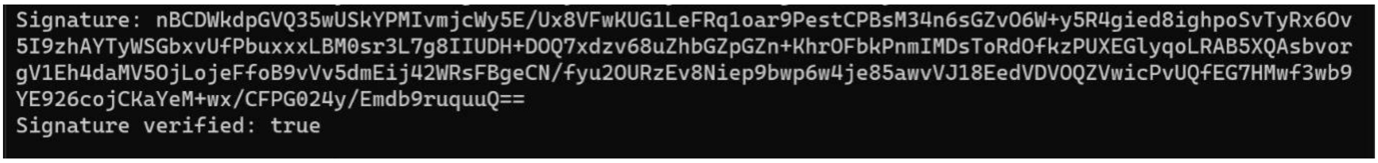


```
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is : 12.0
Decrypted message is : 12
```

Assignment 8

```
import java.security.*;

import java.util.Base64;


public class Assignment8 {

public static void main(String[] args) throws Exception
{ // Generate a key pair

KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA"); keyPairGenerator.initialize(2048);

KeyPair keyPair = keyPairGenerator.generateKeyPair();

// Get the private key

PrivateKey privateKey = keyPair.getPrivate();

// Get the message to be signed

String message = "This is a message to be signed.";

// Create a signature object

Signature signature = Signature.getInstance("SHA256withRSA");

// Initialize the signature object with the private
key signature.initSign(privateKey);

// Add the message to the signature object

signature.update(message.getBytes());

// Calculate the signature

byte[] signatureBytes = signature.sign();

// Save the signature

String signatureString =
Base64.getEncoder().encodeToString(signatureBytes);

System.out.println("Signature: " + signatureString); // Verify the signature

Signature verificationSignature = Signature.getInstance("SHA256withRSA");

// Initialize the verification signature object with the public
key verificationSignature.initVerify(keyPair.getPublic());

// Add the message to the verification signature object

verificationSignature.update(message.getBytes());

// Verify the signature
boolean isVerified =

verificationSignature.verify(signatureBytes);

System.out.println("Signature verified: " + isVerified); }

}
```

Output



```
Signature: nBCDWkdpGVQ35wUSkYPMIvmjcWy5E/Ux8VFwKUG1LeFRq1oar9PestCPBsM34n6sGZvO6W+y5R4gied8ighpoSvTyRx6Ov
5I9zhAYTyWSGbxvUfPbuxxxLBM0sr3L7g8IIUDH+DOQ7xdzv68uZhbGZpGZn+KhrOFbkPnmIMDsToRdOfkzPUXEGlyqoLRAB5XQAsbvor
gV1Eh4daMV5OjLojeFfoB9vVv5dmEij42WRsFBgeCN/fyu2OURzEv8Niep9bwp6w4je85awvVJ18EedVDVOQZVwicPvUQfEG7HMwf3wb9
YE926cojCKaYeM+wx/CFPG024y/Emdb9ruquuQ==
Signature verified: true
```