Amit Kumar Sahu

PRACTICAL ASSIGNMENT- 3

Subject: Information and System Security

18MIS7250 - H Slot

1. Find the GCD using Euclidian algorithm and multiplicative inverse modulo n using Extended-Euclidian algorithm

GCD using Euclidian

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

class gcd_euclidian
{
	public static int gcd(int a, int b)
	{
		if (a == 0)
			return b;

		return gcd(b%a, a);
	}
	public static void main(String[] args)
	{
		Scanner sc= new Scanner(System.in);
		System.out.println("Enter 1st Number");
		int a = sc.nextInt();
		System.out.println("Enter 2nd Number");
		int b = sc.nextInt();
		int g;
		g = gcd(a, b);
		System.out.println("GCD(" + a + " , " + b+ ") = " + g);

	}
}
```

OUTPUT:

```
Enter 1st Number
35
Enter 2nd Number
89
GCD(35 , 89) = 1
```

Multiplicative inverse using Euclidian

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

class multiplicative_inverse_euclidian {

	static void modInverse(int a, int m)
	{
		int g = gcd(a, m);
		if (g != 1)
			System.out.println("Inverse doesn't exist");
		else
		{
			System.out.println(
				"Modular multiplicative inverse is "
				+ power(a, m - 2, m));
		}
	}

	static int power(int x, int y, int m)
	{
		if (y == 0)
			return 1;
		int p = power(x, y / 2, m) % m;
		p = (int)((p * (long)p) % m);
		if (y % 2 == 0)
			return p;
		else
			return (int)((x * (long)p) % m);
	}

	static int gcd(int a, int b)
	{
		if (a == 0)
			return b;
		return gcd(b % a, a);
	}
	public static void main(String args[])
	{
	 Scanner sc = new Scanner(System.in);
	 System.out.println("Enter 1st Number");
	 int a = sc.nextInt();
	 System.out.println("Enter 2nd Number");
	 int m = sc.nextInt();

		modInverse(a, m);
	}
}
```

OUTPUT :

```
Enter 1st Number
72
Enter 2nd Number
6773
Modular multiplicative inverse is 525
```

2. Design a menu based modular arithmetic calculator [addition, subtraction, multiplication, division, inverse of a number (additive and multiplicative)].

```
import java.util.Scanner;

class mod_arithmetic_calc {
  public static void main(String[] args) {

    char operator;
    Double number1, number2, result;

    Scanner input = new Scanner(System.in);

    System.out.println("Choose an operator: \n1. Power in modular arithmetic (x^y
mod p) \n2. Multiplicative Inverse (a mod m) \n3. Addition \n4. Substraction \n5.
Multiplication \n6. Division");
    operator = input.next().charAt(0);

    switch (operator) {


      case '1':
        int res = 1;
          System.out.println("Enter x");
         int x = input.nextInt();
         System.out.println("Enter y");
          int y = input.nextInt();
          System.out.println("Enter p");
          int p = input.nextInt();


          x = x % p;

          if (x == 0) {
             System.out.println("case x is divisible by p");
             break;
          }

          while (y > 0)
          {
             if ((y & 1) != 0)
```

```
                res = (res * x) % p;

            y = y >> 1; // y = y/2
            x = (x * x) % p;
          }
        System.out.println("x^y  mod p is " + res);
        break;


      case '2':
         System.out.println("Enter a");
          int a = input.nextInt();
          System.out.println("Enter m");
           int m = input.nextInt();
           for (int j = 1; j < m; j++)
             if (((a%m) * (j%m)) % m == 1)
                  System.out.println("a inverse mod m is : "+ j);

        break;

      case '3':
         System.out.println("Enter first number");
           number1 = input.nextDouble();

           System.out.println("Enter second number");
           number2 = input.nextDouble();
          result = number1 + number2;
          System.out.println(number1 + " + " + number2 + " = " + result);
          break;

        case '4':
             System.out.println("Enter first number");
            number1 = input.nextDouble();

            System.out.println("Enter second number");
            number2 = input.nextDouble();
          result = number1 - number2;
          System.out.println(number1 + " - " + number2 + " = " + result);
          break;

        case '5':
             System.out.println("Enter first number");
            number1 = input.nextDouble();

            System.out.println("Enter second number");
            number2 = input.nextDouble();
          result = number1 * number2;
          System.out.println(number1 + " * " + number2 + " = " + result);
          break;

        case '6':
             System.out.println("Enter first number");
            number1 = input.nextDouble();

            System.out.println("Enter second number");
```

```
            number2 = input.nextDouble();
          result = number1 / number2;
          System.out.println(number1 + " / " + number2 + " = " + result);
          break;

      default:
        System.out.println("Invalid operator!");
        break;
    }

    input.close();
  }
}
```

OUTPUT:

```
Choose an operator:
1. Power in modular arithmetic (x^y  mod p)
2. Multiplicative Inverse (a mod m)
3. Addition
4. Substraction
5. Multiplication
6. Division
1
Enter x
50
Enter y
100
Enter p
13
x^y  mod p is 3


Choose an operator:
1. Power in modular arithmetic (x^y  mod p)
2. Multiplicative Inverse (a mod m)
3. Addition
4. Substraction
5. Multiplication
6. Division
2
Enter a
42
Enter m
575
a inverse mod m is : 178
```

3. Implement Caesar cipher and multiplicative substitution cipher and try cryptanalyst.

```
import java.util.*;

class ceaser_cipher
```

```
{
    public static void decrypt(String cipher_text)
    {
        char[] letters = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};
        try
        {
            int key = 0;
            while(key != 26)
            {
                String message = "";
                for(int i = 0; i < cipher_text.length(); i ++)
                {
                    int x = cipher_text.charAt(i);
                    message += letters[(x - key) % 26];
                }
                key ++;
                System.out.println("KEY: " + key + "\t" + "MESSAGE: " + message);
            }
        }
        catch(Exception e)
        {
        }
    }
    public static void main(String args[])
    {
        String cipher_text = 
"YQIIKHUOEKYXQTHQJXUHRUJXUVYHIJCQDXUHUJXQDJXUIUSEDTCQDYDHECU";
        decrypt(cipher_text);
    }
}
```

OUTPUT:

```
KEY: 1 MESSAGE: LDVVXUHBRXLKDGUDWKHUEHWKHILUVWPDQKHUHWKDQWKHVHFRQGPDQLQURPH
KEY: 2 MESSAGE: KCUUWTGAQWKJCFTCVJGTDGVJGHKTUVOCPJGTGVJCPVJGUGEQPFOCPKPTQOG
KEY: 3 MESSAGE: JBTTVSFZPVJIBESBUIFSCFUIFGJSTUNBOIFSFUIBOUIFTFDPOENBOJOSPNF
KEY: 4 MESSAGE: IASSUREYOUIHADRATHERBETHEFIRSTMANHERETHANTHESECONDMANINROME
KEY: 5 MESSAGE: HZRRTQDXNTHGZCQZSGDQADSGDEHQRSLZMGDQDSGZMSGDRDBNMCLZMHMQNLD
KEY: 6 MESSAGE: GYQQSPCWMSGFYBPYRFCPZCRFCDGPQRKYLFCPCRFYLRFCQCAMLBKYLGLPMKC
KEY: 7 MESSAGE: FXPPROBVLRFEXAOXQEBOYBQEBCFOPQJXKEBOBQEXKQEBPBZLKAJXKFKOLJB
KEY: 8 MESSAGE: EWOOQNAUKQEDWZNWPDANXAPDABENOPIWJDANAPDWJPDAOAYKJZIWJEJNKIA
KEY: 9 MESSAGE: DVNNPMZTJPDCVYMVOCZMWZOCZADMNOHVICZMZOCVIOCZNZXJIYHVIDIMJHZ
KEY: 10      MESSAGE: CUMMOLYSIOCBUXLUNBYLVYNBYZCLMNGUHBYLYNBUHNBYMYWIHXGUHCHLIGY
KEY: 11      MESSAGE: BTLLNKXRHNBATWKTMAXKUXMAXYBKLMFTGAXKXMATGMAXLXVHGWFTGBGKHFX
KEY: 12      MESSAGE: ASKKMJWQGMAZSVJSLZWJTWLZWXAJKLESFZWJWLZSFLZWKWUGFVESFAFJGEW
KEY: 13      MESSAGE: ZRJJLIVPFLZYRUIRKYVISVKYVWZIJKDREYVIVKYREKYVJVTFEUDREZEIFDV
KEY: 14      MESSAGE: YQIIKHUOEKYXQTHQJXUHRUJXUVYHIJCQDXUHUJXQDJXUIUSEDTCQDYDHECU
KEY: 15      MESSAGE: XPHHJGTNDJXWPSGPIWTGQTIWTUXGHIBPCWTGTIWPCIWTHTRDCSBPCXCGDBT
KEY: 16      MESSAGE: WOGGIFSMCIWVORFOHVSFPSHVSTWFGHAOBVSFSHVOBHVSGSQCBRAOBWBFCAS
KEY: 17      MESSAGE: VNFFHERLBHVUNQENGUREORGURSVEFGZNAURERGUNAGURFRPBAQZNAVAEBZR
KEY: 18      MESSAGE: UMEEGDQKAGUTMPDMFTQDNQFTQRUDEFYMZTQDQFTMZFTQEQOAZPYMZUZDAYQ
KEY: 19      MESSAGE: TLDDFCPJZFTSLOCLESPCMPESPQTCDEXLYSPCPESLYESPDPNZYOXLYTYCZXP
```

```
KEY: 20      MESSAGE: SKCCEBOIYESRKNBKDROBLODROPSBCDWKXROBODRKXDROCOMYXNWKXSXBYWO
KEY: 21      MESSAGE: RJBBDANHXDRQJMAJCQNAKNCQNORABCVJWQNANCQJWCQNBNLXWMVJWRWAXVN
KEY: 22      MESSAGE: QIAACZMGWCQPILZIBPMZJMBPMNQZABUIVPMZMBPIVBPMAMKWVLUIVQVZWUM
KEY: 23      MESSAGE: PHZZBYLFVBPOHKYHAOLYILAOLMPYZATHUOLYLAOHUAOLZLJVUKTHUPUYVTL
KEY: 24      MESSAGE: OGYYAXKEUAONGJXGZNKXHKZNKLOXYZSGTNKXKZNGTZNKYKIUTJSGTOTXUSK
KEY: 25      MESSAGE: NFXXZWJDTZNMFIWFYMJWGJYMJKNWXYRFSMJWJYMFSYMJXJHTSIRFSNSWTRJ
KEY: 26      MESSAGE: MEWWYVICSYMLEHVEXLIVFIXLIJMVWXQERLIVIXLERXLIWIGSRHQERMRVSQI
```

4. Implement Affine cipher and try cryptanalysis.

```java
import java.util.Scanner;
public class affine_cipher

{

    public static String encryptionMessage(String Msg)

    {

        String CTxt = "";

        int a = 3;

        int b = 6;

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

        {

            CTxt = CTxt + (char) ((((a * Msg.charAt(i)) + b) % 26) + 65);

        }

        return CTxt;

    }


    public static String decryptionMessage(String CTxt)

    {

        String Msg = "";

        int a = 3;

        int b = 6;

        int a_inv = 0;

        int flag = 0;

        for (int i = 0; i < 26; i++)
```

```
        {
            flag = (a * i) % 26;
            if (flag == 1)
            {
                a_inv = i;
                System.out.println(i);
            }
        }
        for (int i = 0; i < CTxt.length(); i++)
        {
            Msg = Msg + (char) (((a_inv * ((CTxt.charAt(i) - b)) % 26)) + 65);
        }
        return Msg;
    }

    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the message: ");
        String message = sc.next();
        System.out.println("Message is :" + message);
        System.out.println("Encrypted Message is : "
                + encryptionMessage(message));
        System.out.println("Decrypted Message is: "
                + decryptionMessage(encryptionMessage(message)));
        sc.close();
    }
```

```
}
```

OUTPUT:

```
Enter the message:

Amit
Message is :Amit
Encrypted Message is : TVJQ
9
Decrypted Message is: ASOZ
```

5. Implement Autokey and Playfair ciphers.

Autokey

```
import java.lang.*;
import java.util.*;
import java.util.Scanner;
public class autokey_cipher {

	private static final String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter Message :");
		String msg = sc.nextLine();
		System.out.println("Enter Key :");
		String key = sc.nextLine();

		if (key.matches("[-+]?\\d*\\.?\\d+"))
			key = "" + alphabet.charAt(Integer.parseInt(key));
		String enc = autoEncryption(msg, key);

		System.out.println("Plaintext : " + msg);
		System.out.println("Encrypted : " + enc);
		System.out.println("Decrypted : " + autoDecryption(enc, key));
	}

	public static String autoEncryption(String msg, String key)
	{
		int len = msg.length();


		String newKey = key.concat(msg);
		newKey = newKey.substring(0, newKey.length() - key.length());
		String encryptMsg = "";

		for (int x = 0; x < len; x++) {
			int first = alphabet.indexOf(msg.charAt(x));
```

```
                int second = alphabet.indexOf(newKey.charAt(x));
                int total = (first + second) % 26;
                encryptMsg += alphabet.charAt(total);
            }
            return encryptMsg;
        }

        public static String autoDecryption(String msg, String key)
        {
            String currentKey = key;
            String decryptMsg = "";

            for (int x = 0; x < msg.length(); x++) {
                int get1 = alphabet.indexOf(msg.charAt(x));
                int get2 = alphabet.indexOf(currentKey.charAt(x));
                int total = (get1 - get2) % 26;
                total = (total < 0) ? total + 26 : total;
                decryptMsg += alphabet.charAt(total);
                currentKey += alphabet.charAt(total);
            }
            return decryptMsg;
        }
}
```

OUTPUT:

```
Enter Message :
I LOVE ISS
Enter Key :
N
Plaintext : I LOVE ISS
Encrypted : VHKZJZDHAK
Decrypted : IZLOVEZISS
```

Playfair

```
// Java Program to Enode a Message Using Playfair Cipher

import java.io.*;
import java.util.*;

class Playfair {
        String key;
        String plainText;
        char[][] matrix = new char[5][5];

        public Playfair(String key, String plainText)
        {
            // convert all the characters to lowercase
            this.key = key.toLowerCase();

            this.plainText = plainText.toLowerCase();
        }
```

```java
// function to remove duplicate characters from the key
public void cleanPlayFairKey()
{
	LinkedHashSet<Character> set
		= new LinkedHashSet<Character>();

	String newKey = "";

	for (int i = 0; i < key.length(); i++)
		set.add(key.charAt(i));

	Iterator<Character> it = set.iterator();

	while (it.hasNext())
		newKey += (Character)it.next();

	key = newKey;
}

// function to generate playfair cipher key table
public void generateCipherKey()
{
	Set<Character> set = new HashSet<Character>();

	for (int i = 0; i < key.length(); i++)
	{
		if (key.charAt(i) == 'j')
			continue;
		set.add(key.charAt(i));
	}

	// remove repeated characters from the cipher key
	String tempKey = new String(key);

	for (int i = 0; i < 26; i++)
	{
		char ch = (char)(i + 97);
		if (ch == 'j')
			continue;

		if (!set.contains(ch))
			tempKey += ch;
	}

	// create cipher key table
	for (int i = 0, idx = 0; i < 5; i++)
		for (int j = 0; j < 5; j++)
			matrix[i][j] = tempKey.charAt(idx++);

	System.out.println("Playfair Cipher Key Matrix:");

	for (int i = 0; i < 5; i++)
		System.out.println(Arrays.toString(matrix[i]));
}
```

```
// function to preprocess plaintext
public String formatPlainText()
{
	String message = "";
	int len = plainText.length();

	for (int i = 0; i < len; i++)
	{
		// if plaintext contains the character 'j',
		// replace it with 'i'
		if (plainText.charAt(i) == 'j')
			message += 'i';
		else
			message += plainText.charAt(i);
	}

	// if two consecutive characters are same, then
	// insert character 'x' in between them
	for (int i = 0; i < message.length(); i += 2)
	{
		if (message.charAt(i) == message.charAt(i + 1))
			message = message.substring(0, i + 1) + 'x'
					+ message.substring(i + 1);
	}

	// make the plaintext of even length
	if (len % 2 == 1)
		message += 'x'; // dummy character

	return message;
}

// function to group every two characters
public String[] formPairs(String message)
{
	int len = message.length();
	String[] pairs = new String[len / 2];

	for (int i = 0, cnt = 0; i < len / 2; i++)
		pairs[i] = message.substring(cnt, cnt += 2);

	return pairs;
}

// function to get position of character in key table
public int[] getCharPos(char ch)
{
	int[] keyPos = new int[2];

	for (int i = 0; i < 5; i++)
	{
		for (int j = 0; j < 5; j++)
		{

			if (matrix[i][j] == ch)
```

```
                    {
                            keyPos[0] = i;
                            keyPos[1] = j;
                            break;
                    }
                }
            }
            return keyPos;
        }

        public String encryptMessage()
        {
            String message = formatPlainText();
            String[] msgPairs = formPairs(message);
            String encText = "";

            for (int i = 0; i < msgPairs.length; i++)
            {
                char ch1 = msgPairs[i].charAt(0);
                char ch2 = msgPairs[i].charAt(1);
                int[] ch1Pos = getCharPos(ch1);
                int[] ch2Pos = getCharPos(ch2);

                // if both the characters are in the same row
                if (ch1Pos[0] == ch2Pos[0]) {
                    ch1Pos[1] = (ch1Pos[1] + 1) % 5;
                    ch2Pos[1] = (ch2Pos[1] + 1) % 5;
                }

                // if both the characters are in the same column
                else if (ch1Pos[1] == ch2Pos[1])
                {
                    ch1Pos[0] = (ch1Pos[0] + 1) % 5;
                    ch2Pos[0] = (ch2Pos[0] + 1) % 5;
                }

                // if both the characters are in different rows
                // and columns
                else {
                    int temp = ch1Pos[1];
                    ch1Pos[1] = ch2Pos[1];
                    ch2Pos[1] = temp;
                }

                // get the corresponding cipher characters from
                // the key matrix
                encText = encText + matrix[ch1Pos[0]][ch1Pos[1]]
                            + matrix[ch2Pos[0]][ch2Pos[1]];
            }

            return encText;
        }
}

public class playfair_cipher {
```

```
	public static void main(String[] args)
	{
		String key1 = "crypto";
		String plainText1 = "Corona";

		System.out.println("Key: " + key1);
		System.out.println("PlainText: " + plainText1);

		Playfair pfc1 = new Playfair(key1, plainText1);
		pfc1.cleanPlayFairKey();
		pfc1.generateCipherKey();

		String encText1 = pfc1.encryptMessage();
		System.out.println("Cipher Text is: " + encText1);

		System.out.println("\nExample-2\n");

		String key2 = "Problem";
		String plainText2 = "Hello";

		System.out.println("Key: " + key2);
		System.out.println("PlainText: " + plainText2);

		Playfair pfc2 = new Playfair(key2, plainText2);
		pfc2.cleanPlayFairKey();
		pfc2.generateCipherKey();

		String encText2 = pfc2.encryptMessage();
		System.out.println("Cipher Text is: " + encText2);
	}
}
```

OUTPUT:

```
Key: crypto
PlainText: Corona
Playfair Cipher Key Matrix:
[c, r, y, p, t]
[o, a, b, d, e]
[f, g, h, i, k]
[l, m, n, q, s]
[u, v, w, x, z]
Cipher Text is: ofcamb
```

6. Implement Vigenère cipher and try cryptanalysis.

```
import java.util.*;

class vignere_cipher
{
```

```
static String generateKey(String str, String key)
{
	int x = str.length();

	for (int i = 0; ; i++)
	{
		if (x == i)
			i = 0;
		if (key.length() == str.length())
			break;
		key+=(key.charAt(i));
	}
	return key;
}

// This function returns the encrypted text
// generated with the help of the key
static String cipherText(String str, String key)
{
	String cipher_text="";

	for (int i = 0; i < str.length(); i++)
	{
		// converting in range 0-25
		int x = (str.charAt(i) + key.charAt(i)) %26;

		// convert into alphabets(ASCII)
		x += 'A';

		cipher_text+=(char)(x);
	}
	return cipher_text;
}

// This function decrypts the encrypted text
// and returns the original text
static String originalText(String cipher_text, String key)
{
	String orig_text="";

	for (int i = 0 ; i < cipher_text.length() &&
							i < key.length(); i++)
	{
		// converting in range 0-25
		int x = (cipher_text.charAt(i) -
					key.charAt(i) + 26) %26;

		// convert into alphabets(ASCII)
		x += 'A';
		orig_text+=(char)(x);
	}
	return orig_text;
}

// This function will convert the lower case character to Upper case
```

```java
static String LowerToUpper(String s)
{
	StringBuffer str =new StringBuffer(s);
	for(int i = 0; i < s.length(); i++)
	{
		if(Character.isLowerCase(s.charAt(i)))
		{
			str.setCharAt(i, Character.toUpperCase(s.charAt(i)));
		}
	}
	s = str.toString();
	return s;
}

public static void main(String[] args)
{
	Scanner sc= new Scanner(System.in);
	System.out.println("Enter Plain Text :");
	String Str = sc.nextLine();
	System.out.println("Enter Key :");
	String Keyword = sc.nextLine();

	System.out.println("Plain Text : "+Str);
	System.out.println("Key : "+Keyword);

	String str = LowerToUpper(Str);
	String keyword = LowerToUpper(Keyword);

	String key = generateKey(str, keyword);
	String cipher_text = cipherText(str, key);

	System.out.println("Ciphertext : "
		+ cipher_text + "\n");

	System.out.println("After Decryption : "
		+ originalText(cipher_text, key));
	}
}
```

OUTPUT:

```
Enter Plain Text :
Pandemic
Enter Key :
corona
Plain Text : Pandemic
Key : corona
Ciphertext : ROERRMKQ

After Decryption : PANDEMIC
```

7. Implement Hill cipher and One-time-pad cipher.

Hill Cipher

```
import java.util.*;
class hill_cipher
{
static void getKeyMatrix(String key, int keyMatrix[][])
{
	int k = 0;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			keyMatrix[i][j] = (key.charAt(k)) % 65;
			k++;
		}
	}
}


static void encrypt(int cipherMatrix[][],
			int keyMatrix[][],
			int messageVector[][])
{
	int x, i, j;
	for (i = 0; i < 3; i++)
	{
		for (j = 0; j < 1; j++)
		{
			cipherMatrix[i][j] = 0;

			for (x = 0; x < 3; x++)
			{
				cipherMatrix[i][j] +=
					keyMatrix[i][x] * messageVector[x][j];
			}

			cipherMatrix[i][j] = cipherMatrix[i][j] % 26;
		}
	}
}

static void HillCipher(String message, String key)
{

	int [][]keyMatrix = new int[3][3];
	getKeyMatrix(key, keyMatrix);

	int [][]messageVector = new int[3][1];


	for (int i = 0; i < 3; i++)
		messageVector[i][0] = (message.charAt(i)) % 65;

	int [][]cipherMatrix = new int[3][1];
```

```
        encrypt(cipherMatrix, keyMatrix, messageVector);

        String CipherText="";


        for (int i = 0; i < 3; i++)
              CipherText += (char)(cipherMatrix[i][0] + 65);
        System.out.print(" Ciphertext  is :" + CipherText);
}

public static void main(String[] args)
{
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter the message (3 word) : ");
        String message = sc.nextLine();
        System.out.println("Enter the key : ");
        String key = sc.nextLine();

        HillCipher(message, key);
        }
}
```

OUTPUT:

```
Enter the message (3 word) :
ISS
Enter the key :
UIOEHGTYU
Ciphertext  is :KGI
```

One Time Pad

```
import java.util.*;
import java.io.*;
public class one_time_pad{

        public static String stringEncryption(String text,
                                                    String key)
        {
                String cipherText = "";

                int cipher[] = new int[key.length()];

                for (int i = 0; i < key.length(); i++)
                {
                        cipher[i] = text.charAt(i) - 'A' + key.charAt(i)
                                                    - 'A';
                }
```

```java
        for (int i = 0; i < key.length(); i++)
        {
            if (cipher[i] > 25)
            {
                cipher[i] = cipher[i] - 26;
            }
        }

        for (int i = 0; i < key.length(); i++)
        {
            int x = cipher[i] + 'A';
            cipherText += (char)x;
        }
        return cipherText;
    }


    public static String stringDecryption(String s,String key)
    {
        String plainText = "";
        int plain[] = new int[key.length()];

        for (int i = 0; i < key.length(); i++)
        {
            plain[i]= s.charAt(i) - 'A' - (key.charAt(i) - 'A');
        }

        for (int i = 0; i < key.length(); i++)
        {
            if (plain[i] < 0)
            {
                plain[i] = plain[i] + 26;
            }
        }

        for (int i = 0; i < key.length(); i++)
        {
            int x = plain[i] + 'A';
            plainText += (char)x;
        }


        return plainText;
    }

    public static void main(String[] args)
    {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter Plain Text");
        String plainText = sc.nextLine();
        System.out.println("Enter Key");
        String key = sc.nextLine();

        String encryptedText =
            stringEncryption(plainText.toUpperCase(), key.toUpperCase());
```

```
            System.out.println("Cipher Text - "+ encryptedText);

            System.out.println("Message - "
                        + stringDecryption(encryptedText,
                                                  key.toUpperCase()));
      }
}
```

OUTPUT:

```
Enter Plain Text
Alive
Enter Key
HELLO
Cipher Text - HPTGS
Message - ALIVE
```

8. Implement deterministic and probabilistic Primality testing algorithms

Deterministic – AKS Algorithm

```
import java.util.*;
class AKS_primality_test {

      static long c[] = new long[100];

      static void coef(int n)
      {
            c[0] = 1;
            for (int i = 0; i < n; c[0] = -c[0], i++) {
                  c[1 + i] = 1;

                  for (int j = i; j > 0; j--)
                        c[j] = c[j - 1] - c[j];
            }
      }


      static boolean isPrime(int n)
      {

            coef(n);
            c[0]++;
            c[n]--;

            int i = n;
            while ((i--) > 0 && c[i] % n == 0)
                  ;
            return i < 0;
      }
      public static void main(String[] args)
```

```
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the number you want to test");
                int n = sc.nextInt();
                if (isPrime(n))
                        System.out.println("This is a Prime");
                else
                        System.out.println("Not Prime");
        }
}
```

OUPUT:

```
Enter the number you want to test
37
This is a Prime
```

Probabilistic- Miler-Rabin Primality Test

```
import java.util.Scanner;

import java.util.Random;

import java.math.BigInteger;


public class miller_rabin_prob_primality

{

    public boolean isPrime(long n, int iteration)

    {

        if (n == 0 || n == 1)

            return false;


        if (n == 2)

            return true;


        if (n % 2 == 0)

            return false;

        long s = n - 1;

        while (s % 2 == 0)
```

```java
            s /= 2;

        Random rand = new Random();

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

        {

            long r = Math.abs(rand.nextLong());

            long a = r % (n - 1) + 1, temp = s;

            long mod = modPow(a, temp, n);

            while (temp != n - 1 && mod != 1 && mod != n - 1)

            {

                mod = mulMod(mod, mod, n);

                temp *= 2;

            }

            if (mod != n - 1 && temp % 2 == 0)

                return false;

        }

        return true;

    }

    public long modPow(long a, long b, long c)

    {

        long res = 1;

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

        {

            res *= a;

            res %= c;

        }

        return res % c;

    }
```

```
    public long mulMod(long a, long b, long mod)

    {

        return
BigInteger.valueOf(a).multiply(BigInteger.valueOf(b)).mod(BigInteger.valueOf(mod)).lo
ngValue();

    }

    public static void main (String[] args)
    {

        Scanner scan = new Scanner(System.in);

        System.out.println("Miller Rabin Primality Algorithm Test\n");

        miller_rabin_prob_primality mr = new miller_rabin_prob_primality();


        System.out.println("Enter number\n");

        long num = scan.nextLong();


        System.out.println("\nEnter number of iterations");

        int k = scan.nextInt();

        boolean prime = mr.isPrime(num, k);

        if (prime)

            System.out.println("\n"+ num +" is prime");

        else

            System.out.println("\n"+ num +" is composite");

    }

}
```

OUTPUT:

```
Miller Rabin Primality Algorithm Test

Enter number

2677

Enter number of iterations
78
```

```
2677 is prime
```

9. Implement Chinese Remainder Theorem.

```java
import static java.util.Arrays.stream;

public class CRT {

    public static int chineseRemainder(int[] n, int[] a) {

        int prod = stream(n).reduce(1, (i, j) -> i * j);

        int p, sm = 0;
        for (int i = 0; i < n.length; i++) {
            p = prod / n[i];
            sm += a[i] * mulInv(p, n[i]) * p;
        }
        return sm % prod;
    }

    private static int mulInv(int a, int b) {
        int b0 = b;
        int x0 = 0;
        int x1 = 1;

        if (b == 1)
            return 1;

        while (a > 1) {
            int q = a / b;
            int amb = a % b;
            a = b;
            b = amb;
            int xqx = x1 - q * x0;
            x1 = x0;
            x0 = xqx;
        }

        if (x1 < 0)
            x1 += b0;

        return x1;
    }

    public static void main(String[] args) {
        int[] n = {3, 5, 7};
        int[] a = {2, 3, 2};
        System.out.println(chineseRemainder(n, a));
    }
}
```

OUTPUT :

23

10. Implement RSA cryptosystem

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

public class RSA
{
    public static BigInteger generateE(BigInteger phi)
    {
        BigInteger begin = new BigInteger("3");
        while(begin.compareTo(phi) < 0)
        {
            if(isCoPrime(begin, phi))
                break;
            begin = begin.add(BigInteger.ONE);
            System.out.println(begin);
        }
        return begin;
    }

    public static BigInteger gcd(BigInteger n1, BigInteger n2)
    {
       if(n1 == BigInteger.ZERO)
                    return n2;
             return gcd(n2.mod(n1),n1);
    }

    public static boolean isCoPrime(BigInteger n1, BigInteger n2)
    {
        if(gcd(n1,n2).equals(BigInteger.ONE))
        {
            return true;
        }
        return false;
    }

    public static void main(String[] args)
    {
        BigInteger p, q, n, phi, e, d, m, cipherText, decryptedMessage;
       String plainText;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter p: ");
        p = new BigInteger(sc.nextLine());
        System.out.println("Enter q: ");
        q = new BigInteger(sc.nextLine());
        n = p.multiply(q);
```

```
        phi = (p.subtract(BigInteger.ONE)).multiply((q.subtract(BigInteger.ONE)));
        e = generateE(phi);
        d = e.modInverse(phi);
          System.out.println("Plain Text: ");
          plainText = sc.nextLine();
          m = new BigInteger(plainText.getBytes());
          cipherText = m.modPow(e, n);
          System.out.println("Cipher Text: " + new
String(cipherText.toByteArray()));
          decryptedMessage = cipherText.modPow(d, n);
          System.out.println("Decrypted Message: " + new
String(decryptedMessage.toByteArray()));
    }
}
```

OUTPUT:

```
Enter p:
397
Enter q:
401
4
5
6
7
Plain Text:
NO
Cipher Text: __V
Decrypted Message: NO
```

11. Implement Rabin cryptosystem.

```
import java.util.*;
import java.math.*;
import java.nio.charset.Charset;
import java.security.SecureRandom;

class Cryptography {
      private static Random r = new SecureRandom();
      private static BigInteger TWO = BigInteger.valueOf(2);
      private static BigInteger THREE = BigInteger.valueOf(3);
      private static BigInteger FOUR = BigInteger.valueOf(4);

      public static BigInteger[] generateKey(int bitLength)
      {
            BigInteger p = blumPrime(bitLength / 2);
            BigInteger q = blumPrime(bitLength / 2);
            BigInteger N = p.multiply(q);
            return new BigInteger[] { N, p, q };
      }

      public static BigInteger encrypt(BigInteger m,
```

```
                                                    BigInteger N)
{
        return m.modPow(TWO, N);
}

public static BigInteger[] decrypt(BigInteger c,
                                                    BigInteger p,
                                                    BigInteger q)
{
        BigInteger N = p.multiply(q);
        BigInteger p1 = c.modPow(p
                                                    .add(BigInteger.ONE)
                                                    .divide(FOUR),
                                            p);
        BigInteger p2 = p.subtract(p1);
        BigInteger q1 = c.modPow(q
                                                    .add(BigInteger.ONE)
                                                    .divide(FOUR),
                                            q);
        BigInteger q2 = q.subtract(q1);

        BigInteger[] ext = Gcd(p, q);
        BigInteger y_p = ext[1];
        BigInteger y_q = ext[2];

        BigInteger d1 = y_p.multiply(p)
                                        .multiply(q1)
                                        .add(y_q.multiply(q)
                                                    .multiply(p1))
                                        .mod(N);
        BigInteger d2 = y_p.multiply(p)
                                        .multiply(q2)
                                        .add(y_q.multiply(q)
                                                    .multiply(p1))
                                        .mod(N);
        BigInteger d3 = y_p.multiply(p)
                                        .multiply(q1)
                                        .add(y_q.multiply(q)
                                                    .multiply(p2))
                                        .mod(N);
        BigInteger d4 = y_p.multiply(p)
                                        .multiply(q2)
                                        .add(y_q.multiply(q)
                                                    .multiply(p2))
                                        .mod(N);

        return new BigInteger[] { d1, d2, d3, d4 };
}

public static BigInteger[] Gcd(BigInteger a, BigInteger b)
{
        BigInteger s = BigInteger.ZERO;
        BigInteger old_s = BigInteger.ONE;
        BigInteger t = BigInteger.ONE;
        BigInteger old_t = BigInteger.ZERO;
```

```
		BigInteger r = b;
		BigInteger old_r = a;
		while (!r.equals(BigInteger.ZERO)) {
			BigInteger q = old_r.divide(r);
			BigInteger tr = r;
			r = old_r.subtract(q.multiply(r));
			old_r = tr;

			BigInteger ts = s;
			s = old_s.subtract(q.multiply(s));
			old_s = ts;

			BigInteger tt = t;
			t = old_t.subtract(q.multiply(t));
			old_t = tt;
		}
		return new BigInteger[] { old_r, old_s, old_t };
	}

	public static BigInteger blumPrime(int bitLength)
	{
		BigInteger p;
		do {
			p = BigInteger.probablePrime(bitLength, r);
		} while (!p.mod(FOUR).equals(THREE));
		return p;
	}
}
public class rabin{
	public static void main(String[] args)
	{
		BigInteger[] key = Cryptography.generateKey(512);
		BigInteger n = key[0];
		BigInteger p = key[1];
		BigInteger q = key[2];
		String finalMessage = null;
		int i = 1;
		Scanner sc= new Scanner(System.in);
		System.out.println("Enter you message : ");
		String s = sc.nextLine();

		System.out.println("Message sent by sender : " + s);

		BigInteger m
			= new BigInteger(
				s.getBytes(
					Charset.forName("ascii")));
		BigInteger c = Cryptography.encrypt(m, n);

		System.out.println("Encrypted Message : " + c);

		BigInteger[] m2 = Cryptography.decrypt(c, p, q);
		for (BigInteger b : m2) {
			String dec = new String(
				b.toByteArray(),
```

```
                        Charset.forName("ascii"));
                if (dec.equals(s)) {
                        finalMessage = dec;
                }
                i++;
            }
            System.out.println(
                "Message received by Receiver : "
                + finalMessage);
        }
}
```

OUTPUT:

```
Enter you message :
The pandemic is devastating
Message sent by sender : The pandemic is devastating
Encrypted Message :
1205711177103860593587073086930528288252473857610269862323684790201476161220480242144
035061432386535110187941108883950763020823921
Message received by Receiver : The pandemic is devastating
```

12. Implement ElGamal cryptosystem.

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

public class ElGamal {

        public static void main(String[] args) {

                Scanner stdin = new Scanner(System.in);
                Random r = new Random();
                System.out.println("Enter the approximate value of the prime number for
your El Gamal key.");
                BigInteger p = getNextPrime(stdin.next());

                BigInteger g = getGenerator(p, r);

                if (g != null) {

                        BigInteger a = new BigInteger(p.bitLength()+1, r);
                        a = a.mod(p);
                        if (a.equals(BigInteger.ZERO) || a.equals(BigInteger.ONE))
                                a = a.add(new BigInteger("2"));

                        BigInteger b = g.modPow(a, p);

                        System.out.println("Post p = "+p+" g = "+g+" b = "+b);


                        BigInteger k = new BigInteger(p.bitLength()+1, r);
```

```
                        k = k.mod(p);
                        if (k.equals(BigInteger.ZERO) || k.equals(BigInteger.ONE))
                              k = k.add(new BigInteger("2"));


                        BigInteger c1 = g.modPow(k, p);
                        BigInteger c2 = b.modPow(k, p);

                        // Here we get the message from the user.
                        System.out.println("Please enter your message. It should be in
between 1 and "+p);
                        BigInteger m = new BigInteger(stdin.next());

                        c2 = c2.multiply(m);
                        c2 = c2.mod(p);

                        System.out.println("The corresponding cipher texts are c1 =
"+c1+" c2 = "+c2);

                        BigInteger temp = c1.modPow(a,p);
                        temp = temp.modInverse(p);


                        System.out.println("Here is c1^ -a = "+temp);

                        BigInteger recover = temp.multiply(c2);
                        recover = recover.mod(p);

                        System.out.println("The original message = "+recover);
                  }

                  else
                        System.out.println("Sorry, a generator for your prime couldn't be
found.");

            }


            // Incrementally tries each BigInteger starting at the value passed
            // in as a parameter until one of them is tests as being prime.
            public static BigInteger getNextPrime(String ans) {
                  BigInteger one = new BigInteger("1");
                  BigInteger test = new BigInteger(ans);
                  while (!test.isProbablePrime(99))
                        test = test.add(one);
                  return test;
            }


            public static BigInteger getGenerator(BigInteger p, Random r) {

                  int numtries = 0;

                  while (numtries < 1000) {
```

```
            BigInteger rand = new BigInteger(p.bitLength()+1,r);
                  rand = rand.mod(p);

            BigInteger exp = BigInteger.ONE;
            BigInteger next = rand;

                  if (next.equals(BigInteger.ZERO) || next.equals(BigInteger.ONE))
                        continue;

            while (!next.equals(BigInteger.ONE)) {
                  next = (next.multiply(rand)).mod(p);
                  exp = exp.add(BigInteger.ONE);
            }


            if (exp.equals(p.subtract(BigInteger.ONE)))
                  return rand;

                  numtries++;
      }

      return null;

  }

}
```

OUTPUT:

```
Enter the approximate value of the prime number for your El Gamal key.
343
Post p = 347 g = 193 b = 25
Please enter your message. It should be in between 1 and 347
345
The corresponding cipher texts are c1 = 36 c2 = 13
Here is c1^ -a = 160
The original message = 345
```

13. Implement RSA Digital Signature Scheme.

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

import javax.xml.bind.DatatypeConverter;
      public class RSA
      {
            private int keyLength = 1024;
            private SecureRandom     rand;
         private BigInteger p;
         private BigInteger q;
         private BigInteger n;
         private BigInteger lambda;
```

```
        private BigInteger e;
        private BigInteger d;
        private BigInteger[] publicKey = new BigInteger[2];
        private BigInteger[] privateKey = new BigInteger[2];
        private final BigInteger one = new BigInteger("1");

        public RSA()
        {
            rand = new SecureRandom();
            p = BigInteger.probablePrime(keyLength, rand);
            q = BigInteger.probablePrime(keyLength, rand);
            n = p.multiply(q);
            lambda = p.subtract(one).multiply(q.subtract(one));
            e = BigInteger.probablePrime(keyLength / 2, rand);
            while (e.intValue()<lambda.intValue() && (lambda.gcd(e).intValue()-
one.intValue()) > 0){
                e.add(one);
            }
            d = e.modInverse(lambda);
            publicKey[0]=e;
            publicKey[1]=n;

            privateKey[0]=d;
            privateKey[1]=n;

        }
        public byte[] encrypt(BigInteger d, BigInteger n, byte[] message)
        {
            return (new BigInteger(message)).modPow(d, n).toByteArray();
        }

        public byte[] decrypt(BigInteger e, BigInteger n, byte[] message)
        {
            return (new BigInteger(message)).modPow(e, n).toByteArray();
        }

        public String toString(byte[] cipher)
        {
            String s = "";
            for (byte b : cipher)    {
                s += Byte.toString(b);
            }
            return s;
        }

        public BigInteger[] getPublicKey() {
                return publicKey;
          }

          public BigInteger[] getPrivateKey() {
                return privateKey;
          }

    }
          final public class SHA256 {
```

```java
        //Hash values:
        private int[] H = {
                0x6a09e667,
                0xbb67ae85,
                0x3c6ef372,
                0xa54ff53a,
                0x510e527f,
                0x9b05688c,
                0x1f83d9ab,
                0x5be0cd19
        };
        //Constants:
        private static int[] K = {
                0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
                0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
                0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
                0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
                0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
                0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
                0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
                0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
                0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
                0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
                0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
                0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
                0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
                0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
                0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
                0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
        };

        //Array of 64 byte/512 bit chunks:
        private final static byte[] chunk = new byte[64];

        //a 64-entry message schedule array w[0..63] of 32-bit words:
        private final static int[] w = new int[64];

        private static byte[] preprocess(byte[] msgBytes) {
            int length = msgBytes.length;
            long bits = length*8;
            int msgLength = (length%64)*8; //length modulo 64 in bits
            int padLength;
            //since the first bit in the padding is set to 1, we need
padding bigger than 1 byte.
            //pad length is 512 - L = 1+k+64
            if ((512 - msgLength) > 8) {
                padLength = 512 - msgLength;
            } else {
                padLength = 1024 - msgLength;
            }
          //all bits in the pad are set to 0:
            byte[] pad = new byte[padLength/8];
          //append 1 bit to the pad:
            pad[0] = (byte) 0x80;
          //Length of message in 64 bits is appended (in Big Endian):
```

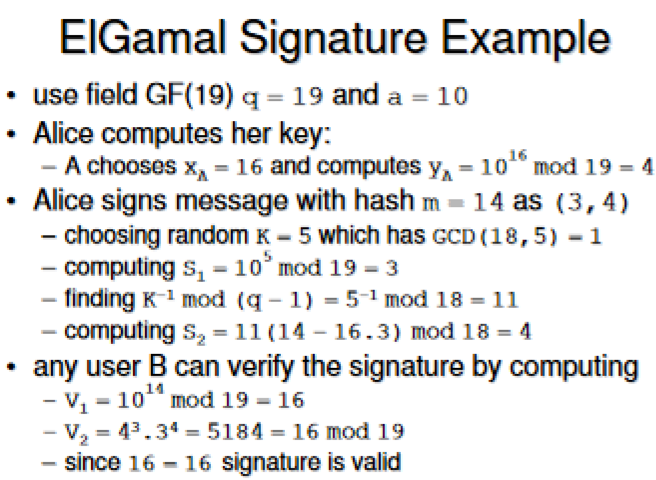
```
                    for (int i = 0; i < 8; i++) {
                        pad[pad.length - 1 - i] = (byte) (0xFF&(bits >>> (8 *
i)));
                    }
                  //k bits are still 0.
                    byte[] result = new byte[length + padLength/8];
                    for (int i = 0; i<length; i++) {
                        result[i]=msgBytes[i];
                    }
                    for (int i = length; i<pad.length+length; i++) {
                        result[i]=pad[i-length];
                    }
                    return result;
                }

                private static void msgProcessing() {
                    for (int i=0; i<16; i++) {
                        w[i] = 0;
                        for (int j = 0; j<4; j++) {
                            w[i] |= ((0x000000FF&chunk[j+4*i]) << (24-j*8));
                        }
                    }
                    for (int i=16; i<64; i++) {
                         w[i]=0;
                        int s0 = Integer.rotateRight(w[i-15],7) ^
Integer.rotateRight(w[i-15],18) ^
                                (w[i - 15] >>> 3);
                        int s1 = Integer.rotateRight(w[i-2],17) ^
Integer.rotateRight(w[i-2],19) ^
                                (w[i-2] >>> 10);
                        w[i] = w[i-16] + s0 + w[i-7] + s1;
                    }
                }

                private static void update(int[] h, int[] w, int j) {
                   int S1 = Integer.rotateRight(h[4], 6) ^
                            Integer.rotateRight(h[4], 11) ^
                            Integer.rotateRight(h[4], 25);
                   int ch = (h[4] & h[5]) ^ (~h[4] & h[6]);
                   int temp1 = h[7] + S1 + ch + K[j] + w[j];

                    int S0 = Integer.rotateRight(h[0], 2) ^
                            Integer.rotateRight(h[0], 13) ^
                            Integer.rotateRight(h[0], 22);
                    int maj = (h[0] & h[1]) ^ (h[0] & h[2]) ^ (h[1] &
                              h[2]);
                    int temp2 = S0 + maj;

                    h[7] = h[6];
                    h[6] = h[5];
                    h[5] = h[4];
                    h[4] = h[3] + temp1;
                    h[3] = h[2];
                    h[2] = h[1];
                    h[1] = h[0];
```

```
            h[0] = temp1 + temp2;
        }

        public static byte[] toByteArray(int input) {
            byte[] output = new byte[4];
            for (int i = 0; i < 4; i++) {
                output[i] = (byte) (0xff&(input >>> (56-i*8)));
            }
            return output;
        }

        public byte[] hash(byte[] msgBytes) {
            byte[] processed = preprocess(msgBytes);
            int[] hVals = new int[8];
            for (int i = 0; i<8; i++) {
                 hVals[i]=H[i];
            }
            for (int i = 0; i < processed.length / 64; ++i) {
                 for (int j = 0; j<processed.length; j++) {
                        chunk[j]=processed[64*i+j];
                 }
                msgProcessing();
                int[] temp = new int[8];
                for (int j = 0; j < 8; j++) {
                 temp[j] = hVals[j];
                }
                for (int j = 0; j < 64; ++j) {
                    update(temp, w, j);
                }
                for (int j = 0; j < 8; ++j) {
                    hVals[j] += temp[j];
                }
            }
            byte[] finalHash = new byte[32];
            for (int i = 0; i < 8; i++) {
                 for (int j = 0; j < 4; j++) {
                        finalHash[4*i+j]=toByteArray(hVals[i])[j];
                 }
            }
            return finalHash;
        }
    }
    public class Main {

            public static void main(String[] args) {
                 //Step 4 in the signing process and step 1, 2 and 4 in the
verification process is not
                 //necessary, but I am only going through them for
simulation purposes.
                 //Signing process:
                 //1. Bob generates a message (user input)
                 @SuppressWarnings("resource")
                 Scanner inn = new Scanner(System.in);
                 String message = inn.nextLine();
```

```
                    //2. Bob Generates hash value for the message (SHA-256
with arbitrary length input)
                    SHA256 sha256 = new SHA256();
                byte[] msgBytes = message.getBytes();
                byte[] bobsHash = sha256.hash(msgBytes);
                String bobsHashStr =
DatatypeConverter.printHexBinary(bobsHash);//variable for printing
                System.out.println("Bob's hash in String: " + bobsHashStr);

                    //3. Bob uses the hash value together with a private key
as input for RSA algorithm
                RSA rsa = new RSA();
                BigInteger[] pk = rsa.getPrivateKey();//Bob's private key
                BigInteger[] puk = rsa.getPublicKey();//Bob's public key
                System.out.println("Encrypting Bob's hash: " + bobsHashStr);
                System.out.println("Bob's hash in Bytes: "
                        + rsa.toString(bobsHashStr.getBytes()));
                //Encrypt/Generate digital signature
                byte[] encrypted =
rsa.encrypt(pk[0],pk[1],bobsHashStr.getBytes());
                System.out.println("Bob's encrypted Hash: " +
DatatypeConverter.printHexBinary(encrypted));

                    //4. Bob sends message with RSA signature attached.

                    /*verification:
                    1. Alice receives the message with signature.*/
                String receivedMessage = message;

                    /*2. Alice calculates a hash value for the received
message, just as Bob did in step 2 of
                the signing process (SHA-256 with arbitrary length input).*/
                byte[] rMsgBytes = receivedMessage.getBytes();
                byte[] rHash = sha256.hash(rMsgBytes);
                String rHashStr =
DatatypeConverter.printHexBinary(rHash);//variable for printing
                System.out.println("Alices' hash in String: " + rHashStr);

                    /*3. Alice provides the received encryption and Bob's
public key as input for RSA
                decryption.*/
                byte[] decrypted = rsa.decrypt(puk[0],puk[1],encrypted);
                System.out.println("Alices' decrypted Hash in bytes: " +
rsa.toString(decrypted));
                System.out.println("Alices' decrypted Hash in String: " + new
String(decrypted));

                /*4. Alice compares the RSA decryption with her calculated hash
value for the received
                message. If the algoritm returns the result that the signature
is valid (which in this
                case it is, because Alice's decrypted hash matches Bob's
original hash), Alice is assured
                that the message has been signed by Bob, because nobody else has
his private key, so
```

```
                    nobody else could have created a signature for this message that
could be verified for
                    this message using Bob's key.*/
                    if (new String(decrypted).equals(rHashStr)) {
                         System.out.println("Verification successfull! Alices'
decryption matches her "
                                     + "calculated hash -> The message is signed
by Bob!");
                    } else {
                         System.out.println("Verification successful! Alice's
decryption does not match her "
                                     + "calculated hash -> The message is NOT
signed by Bob!");
                    }

               }

          }
```

14. Implement ElGamal Digital Signature Scheme.




```
import java.util.Scanner;

public class Elgamal_digital_sign {
     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          System.out.println("Enter the value of p");
          long p=sc.nextLong();
          System.out.println("Enter the value of alpha");
          long alpha=sc.nextLong();
          System.out.println("Enter the value of m");
          long m=sc.nextLong();
          System.out.println("Enter the value of k");
          long k=sc.nextLong();
          signAlgo sign = new signAlgo(p,alpha,m,k);
```

```
        verify v = new verify(sign.p,sign.alpha,sign.beta,sign.m,sign.r,sign.s);
        v.verified();
    }
}



class signAlgo {

    public long p,alpha,beta,m,r,s,k;

    private long z=16;

    signAlgo(long a, long b,long c,long d){
        p=a;
        alpha=b;
        beta=((long)Math.pow(alpha, z))%p;
        m=c;
        k=d;//createK();
        r=createR(alpha,k);
        s=createS();
    }

    long gcd(long a,long b) {
        if(a<b)
            return gcd(b,a);
        else if(a%b==0)
            return b;
        else
            return gcd(b,a%b);
    }

    /*long createK() {
        long a = 2*(p-1);
        while(gcd(a,p-1)!=1) {
            a = (long) (Math.random()*(p-1));
        }
        return a;
    }*/

    long invK() {
        for (int x = 1; x < p-1; x++)
            if ((k * x) % (p-1) == 1)
              return x;
         return 1;
    }

    long createR(long b, long c) {
        long a=((long)Math.pow(b, c));
        if(a<(long)Math.pow(2, 36)-1)
            return a%p;
        else//(a==(long)Math.pow(2, 36)-1)
            return (createR(b,c/2)*createR(b,c-c/2))%p;
    }
```

```
	long createS() {
		long a=(invK()*(m-z*r))%(p-1);
		if(a>=0)
			return a;
		else
			return (a+p-1);
	}


}



class verify {

	public long p,alpha,beta,m,r,s;

	verify(long a, long b,long c,long d, long e,long f){
		p=a;
		alpha=b;
		beta=c;
		m=d;
		r=e;
		s=f;
	}

	/*long v1() {
		return ((((long)Math.pow(beta,r))*((long)(Math.pow(r, s))))%p);
	}*/

	long v1(long b, long c,long d,long e) {
		long a=(((long)Math.pow(b, c))*((long)(Math.pow(d, e))));
		if(a<(long)Math.pow(2, 36)-1)
			return a%p;
		else//(a==(long)Math.pow(2, 36)-1)
			return (v1(b,c/2,d,e/2)*v1(b,c-c/2,d,e-e/2))%p;
	}

	long v2() {
		return (((long)Math.pow(alpha,m))%p);
	}

	long v2(long b, long c) {
		long a=((long)Math.pow(b, c));
		if(a<(long)Math.pow(2, 36)-1)
			return a%p;
		else//(a==(long)Math.pow(2, 36)-1)
			return (v2(b,c/2)*v2(b,c-c/2))%p;
	}

	void verified() {
		if(v1(beta,r,r,s)==v2(alpha,m)) {
			System.out.println("Signature verified using ElGamal.");
			System.out.println("The value of v1 mod p: "+v1(beta,r,r,s));
```

```
                System.out.println("The value of v2 mod p: "+v2(alpha,m));
            }
            else {
                System.out.println("Signature missmatch");
                System.out.println("The value of v1 mod p: "+v1(beta,r,r,s));
                System.out.println("The value of v2 mod p: "+v2(alpha,m));
            }
        }
}
```

OUTPUT:

```
Enter the value of p
19
Enter the value of alpha
10
Enter the value of m
14
Enter the value of k
5
Signature verified using ElGamal.
The value of v1 mod p: 16
The value of v2 mod p: 16
```

15. Implement Diffie-Hellman Key-Exchange Algorithm.

```
import java.util.*;
import java.math.*;
import java.lang.*;

public class Diffi_hellman
{
    public static boolean isPrime(Long number)
    {
        for(Long i = 2L; i <= number / 2; i ++)
        {
            Long temp= number % i;
            if(temp == 0)
            {
                return false;
            }
        }
        return true;
    }
```

```
public static Long generateRandom(Long min, Long max)
{
    Random rand = new Random();
    Long randomNumber = min + ((long)(rand.nextDouble() * (max - min)));
    return randomNumber;
}

public static Long generateRandomPrime(Long min, Long max)
{
    boolean prime = true;
    Long randomNumber = 0L;

    while(prime)
    {
        randomNumber = generateRandom(min, max);
        prime = !isPrime(randomNumber);
    }
    return randomNumber;
}

public static Long findPrimitiveRoot(Long p)
{
    Long k, o = 1L;
    for(Long i = 2L; i < p; i ++)
    {
        k = (long) Math.pow((double) i, (double) o);
        k %= p;
        while(k > 1)
        {
            o ++;
            k *= i;
            k %= p;
        }
        if(o == p - 1)
        {
            return i;
        }
        o = 1L;
    }
    return -1L;
}

public static Long calculateMod(Long base, Long exp, Long p)
{
    return ((long) Math.pow((double)base, (double)exp)) % p;
}

public static void main(String[] args)
{
    try
    {
        Long p, a, m, n;
        Long X, Y;
        Long K1, K2;
        Scanner sc = new Scanner(System.in);
```

```
            p = generateRandomPrime(1000L, 100000L);
            a = findPrimitiveRoot(p);
            m = generateRandomPrime(10L, 50L);
            n = generateRandomPrime(10L, 50L);
            X = calculateMod(a, m, p);
            Y = calculateMod(a, n, p);
            K1 = calculateMod(Y, m, p);
            K2 = calculateMod(X, n, p);
            System.out.println("p: " + p + " " + "a: " + a);
            System.out.println("m: " + m + " " + "n: " + n);
            System.out.println("X: " + X + " " + "Y: " + Y);
            System.out.println("K1: " + K1 + " " + "K2: " + K2);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

OUTPUT:

```
p: 47657 a: 3
m: 23 n: 19
X: 16003 Y: 2551
K1: 39055 K2: 39055
```