Thakur Educational Trust's (Regd.)
THAKUR COLLEGE OF SCIENCE & COMMERCE
AUTONOMOUS COLLEGE, PERMANENTLY AFFILIATED TO UNIVERSITY OF MUMBAI
NAAC Accredited Grade 'A' (3rd Cycle) & ISO 9001: 2015 (Certified)
Best College Award by University of Mumbai for the Year 2018-2019
CELEBRATING 25 YEARS OF GLORY

M.Sc C.S - II
SEM IV
Journal

Roll No.	4733
Name	Neeraj Venkatsai Laxminarayanrao Appari
Subject	Cyber Security

Thakur Educational Trust's (Regd.)

THAKUR COLLEGE OF SCIENCE & COMMERCE

AUTONOMOUS COLLEGE, PERMANENTLY AFFILIATED TO UNIVERSITY OF MUMBAI

NAAC Accredited Grade 'A' (3rd Cycle) & ISO 9001: 2015 (Certified)

Best College Award by University of Mumbai for the Year 2018-2019

CELEBRATING 25 YEARS OF GLORY

Department of Computer Science

Journal Certificate

Class: M.Sc. Computer Science – Part II (Semester: IV)

Roll No: **4733**

Academic Year: 2023-2024

This is to certify that the work entered in this journal is the work of Ms. **Neeraj Venkatsai Laxminarayanrao Appari**, who has worked for academic year 2023-2024 in the computer laboratory. He / She has completed prescribed practical of following course satisfactorily.

Course Title:- **Cyber & Information Security (Cryptography and Crypt Analysis)**

Teacher Incharge **Head of Department**

Date: **Examiner**

INDEX

Practical No.:1

Aim: Write a program to implement following:

• Chinese Reminder Theorem

• Fermat's Little Theorem

Theory:

- **Chinese Remainder Theorem**

We are given two arrays num[0..k-1] and rem[0..k-1]. In num[0..k-1], every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

x % num[0] = rem[0],

x % num[1] = rem[1],

....

x % num[k-1] = rem[k-1]

Basically, we are given k numbers which are pairwise coprime, and given remainders of these numbers when an unknown number x is divided by them. We need to find the minimum possible value of x that produces given remainders

Chinese Remainder Theorem states that there always exists an x that satisfies given congruences.

Let num[0], num[1], …num[k-1] be positive integers that are pairwise coprime. Then, for any given sequence of integers rem[0], rem[1], … rem[k-1], there exists an integer x solving the following system of simultaneous congruences.

$$\begin{cases} x \equiv rem[0] & \pmod{num[0]} \\ \quad \ldots & \\ x \equiv rem[k-1] & \pmod{num[k-1]} \end{cases}$$

Furthermore, all solutions x of this system are congruent modulo the product, prod = num[0] * num[1] * ... * nun[k-1]. Hence

$$x \equiv y \pmod{num[i]}, \quad 0 \le i \le k-1 \qquad \Longleftrightarrow \qquad x \equiv y \pmod{prod}.$$

Fermat's Little Theorem

Fermat's little theorem states that if p is a prime number, then for any integer a, the number $a^{p} - a$ is an integer multiple of p.

Here p is a prime number

$a^p \equiv a \pmod{p}$.

Special Case: If a is not divisible by p, Fermat's little theorem is equivalent to the statement that $a^{p-1}-1$ is an integer multiple of p.

$a^{p-1} \equiv 1 \pmod{p}$

OR

$a^{p-1} \% p = 1$
Here a is not divisible by p.

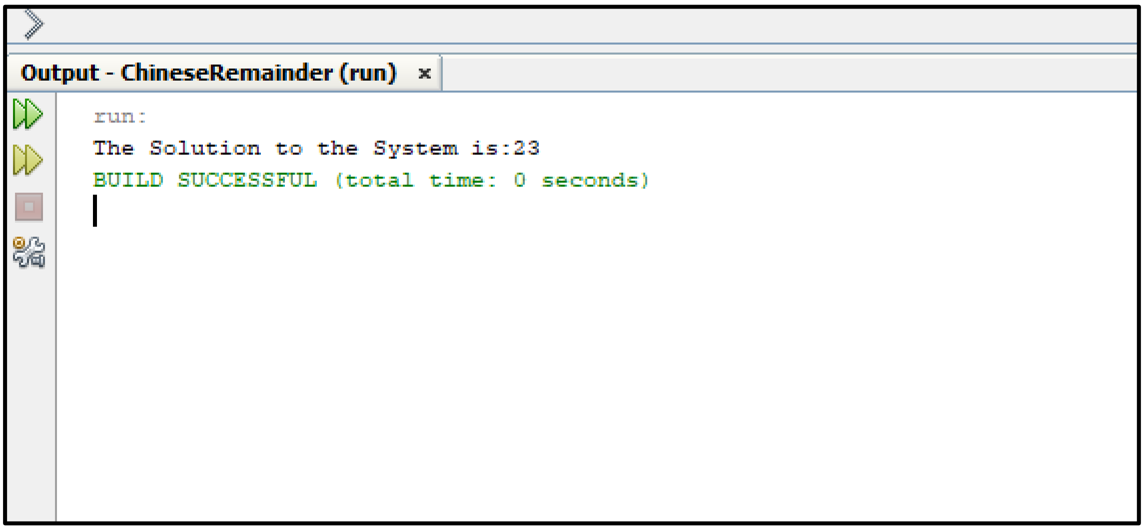
Code :

```
package chineseremainder;

import static java.util.Arrays.stream;
public class ChineseRemainder
{
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("The Solution to the System is:"+ chineseRemainder(n, a));
}
}
```

Output:



- **Fermat Little theorem**

Code:

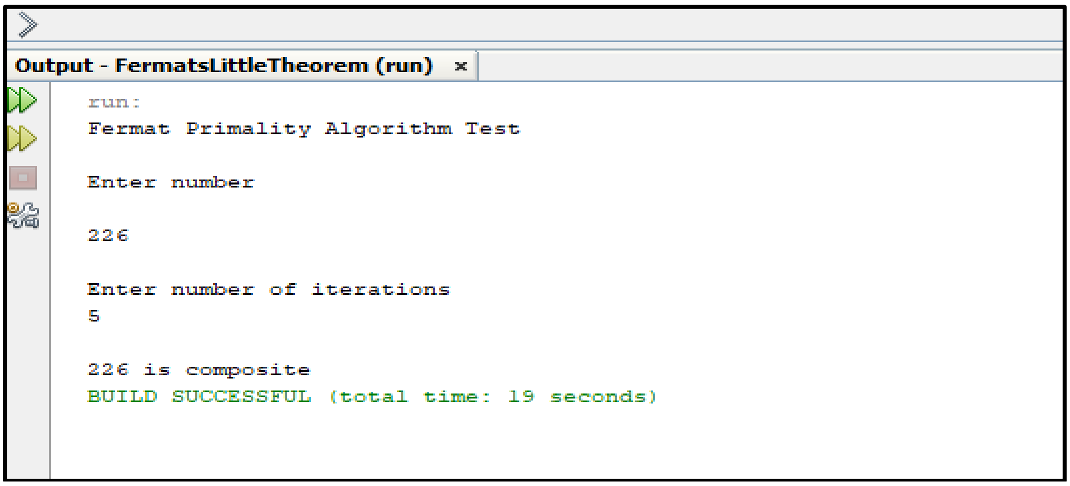
```
package fermatslittletheorem;
import java.util.*;
public class FermatsLittleTheorem

{
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;
Random rand = new Random();
for(int i=0; i<iteration; i++)
{
long r = Math.abs(rand.nextLong());
long a = r % (n - 1) + 1;
if(modPow(a, n - 1, n) != 1)
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 static void main (String[] args)
{
Scanner scan = new Scanner(System.in);
System.out.println("Fermat Primality Algorithm Test\n");
FermatsLittleTheorem fl = new FermatsLittleTheorem();
System.out.println("Enter number\n");
long num = scan.nextLong();
System.out.println("\nEnter number of iterations");
int k = scan.nextInt();
boolean prime = fl.isPrime(num, k);
if(prime)

System.out.println("\n" + num + " is prime");
else
System.out.println("\n" + num + " is composite");
}
}
```

Output:



Conclusion:

The implementation of Chinese remainder theorem and Fermat’s Little Theorem demonstrates the theorem's practicality and efficiency in finding solutions for systems of modular equations.

Practical No.:2

Aim: Write a program to implement following:

❖ Affine Cipher
❖ Rail Fence Technique
❖ Simple Columnar Technique
❖ Vermin Cipher
❖ Hill Cipher to perform encrption and decryption.

Theory:

AFFINE CIPHER

The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which.
The whole process relies on working modulo m (the length of the alphabet used). In the affine cipher, the letters of an alphabet of size m are first mapped to the integers in the range 0 … m-1.

The 'key' for the Affine cipher consists of 2 numbers, we'll call them a and b. The following discussion assumes the use of a 26 character alphabet (m = 26). a should be chosen to be relatively prime to m (i.e. a should have no factors in common with m).

E (x) = (a x + b) mod m

modulus m: size of the alphabet

a and b: key of the cipher.

a must be chosen such that a and m are coprime.

<u>Decryption</u>
In deciphering the ciphertext, we must perform the opposite (or inverse) functions on the ciphertext to retrieve the plaintext. Once again, the first step is to convert each of the ciphertext letters into their integer values. The decryption function is

D (x) = a^-1 (x - b) mod m

a^-1 : modular multiplicative inverse of a modulo m. i.e., it satisfies the equation

1 = a a^-1 mod m .

RAIlFENCE CIPHER

Given a plain-text message and a numeric key, cipher/de-cipher the given text using Rail Fence algorithm.

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

Encryption
Input : "attack at once"
Key = 2
Output : atc toctaka ne
Decryption
Input : "atc toctaka ne"
Key = 2
Output : attack at once

FIG

* _ _ _ * _ _ _ * _ _ _ *
_ * _ * _ * _ * _ * _ *
_ _ * _ _ _ * _ _ _ * _

SIMPLE COLUMNAR TECHNIQUE

Given a plain-text message and a numeric key, cipher/de-cipher the given text using Columnar Transposition Cipher The Columnar Transposition Cipher is a form of transposition cipher just like
<u>Rail Fence Cipher</u>
. Columnar Transposition involves writing the plaintext out in rows, and then reading the cipher text off in columns one by one.
Encryption
Input : Geeks on work
Key = HACK
Output : e w_eoo_Gs kknr_
Decryption
Input : e w_eoo_Gs kknr_
Key = HACK
Output : Geeks on work

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”

VERNAM CIPHER
Vernam Cipher is a method of encrypting alphabetic text. It is one of the Substitution techniques for converting plain text into cipher text. In this mechanism, we assign a number to each character of the Plain-Text, like

(a=0,b=1,c=2,…z=25).
Method to take key: In the Vernam cipher algorithm, we take a key to encrypt the plain text whose length should be equal to the length of the plain text.
Encryption Algorithm

- Assign a number to each character of the plain text and the key according to alphabetical order.
- Bitwise XOR both the number (Corresponding plain-text character number and Key character number).
- Subtract the number from 26 if the resulting number is greater than or equal to 26, if it isn't then leave it.

Example 1:
Plain-Text: O A K
Key: S O N
O ==> 14 = 0 1 1 1 0
S ==> 18 = 1 0 0 1 0
Bitwise XOR Result: 1 1 1 0 0 = 28
Since the resulting number is greater than 26, subtract 26 from it. Then convert the Cipher-Text character number to the Cipher-Text character.
28 - 26 = 2 ==> C
CIPHER-TEXT: C
Similarly, do the same for the other corresponding characters,

❖ Affine Cipher

Source Code:-

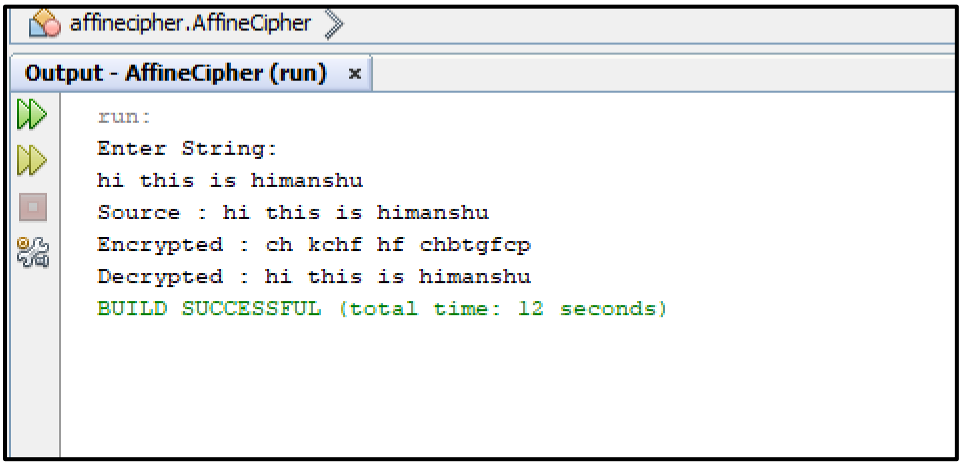
```
package affinecipher;

import java.math.*;
import java.util.*;
public class AffineCipher
{
private static int firstKey = 5;
private static int secondKey = 19;
private static int module = 26;
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter String:");
String input = sc.nextLine();
String cipher = encrypt(input);
String decipher = decrypt(cipher);
System.out.println("Source : " + input);
System.out.println("Encrypted : " + cipher);
System.out.println("Decrypted : " + decipher);
}
static String encrypt(String input)
{
StringBuilder builder = new StringBuilder();
```

```
for(int in = 0; in<input.length(); in++)
{
char character = input.charAt(in);
if(Character.isLetter(character))
{
character = (char) ((firstKey * (character - 'a') + secondKey) %
module + 'a');
}
builder.append(character);
}
return builder.toString();

}
static String decrypt(String input)
{
StringBuilder builder = new StringBuilder();
BigInteger inverse =
BigInteger.valueOf(firstKey).modInverse(BigInteger.valueOf(module));
for(int in=0;in<input.length();in++)
{
char character = input.charAt(in);
if(Character.isLetter(character))
{
int decoded = inverse.intValue() * (character - 'a' - secondKey +
module);
character = (char)(decoded % module + 'a');
}
builder.append(character);
}
return builder.toString();
}
}
```

Output:



- **Railfence Cipher**

Code:

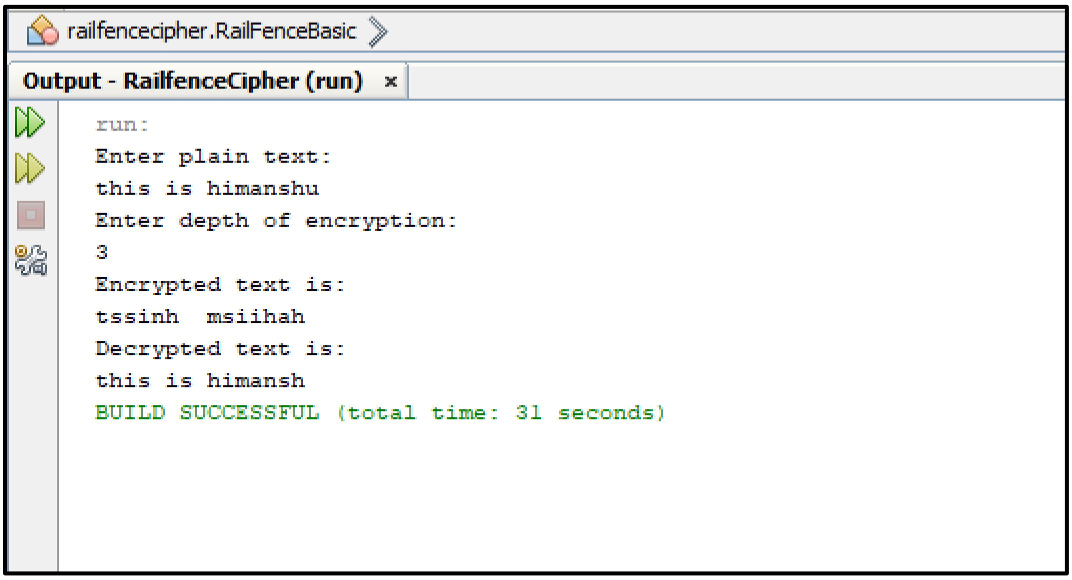
```
package railfencecipher;
import java.util.*;
public class RailfenceCipher
{
public static void main(String[] args) throws Exception
{
RailFenceBasic rf = new RailFenceBasic();

Scanner scn = new Scanner(System.in);
int depth;
String plainText, cipherText, decryptedText;
System.out.println("Enter plain text:");
plainText = scn.nextLine();
System.out.println("Enter depth of encryption:");
depth = scn.nextInt();
cipherText = rf.Encryption(plainText,depth);
System.out.println("Encrypted text is:\n"+cipherText);
decryptedText = rf.Decryption(cipherText, depth);
System.out.println("Decrypted text is:\n"+decryptedText);
}
}
class RailFenceBasic
{
int depth;
String Encryption(String plainText,int depth) throws Exception
{
int r = depth, len = plainText.length();
int c = len/depth;
char mat[][] = new char[r][c];
int k = 0;
String cipherText = "";
for(int i=0;i<c;i++)
{
for(int j=0;j<r;j++)
{
if(k!=len)
mat[j][i] = plainText.charAt(k++);
else
mat[j][i]='X';
}
}
for(int i=0;i<r;i++)
{
```

```
for(int j=0;j<c;j++)
{
cipherText += mat[i][j];
}
}
return cipherText;
}

String Decryption(String cipherText,int depth)throws Exception
{
int r = depth,len=cipherText.length();
int c = len/depth;
char mat[][] = new char[r][c];
int k = 0;
String plainText = "";
for(int i=0;i<r;i++)
{
for(int j=0;j<c;j++)
{
mat[i][j] = cipherText.charAt(k++);
}
}
for(int i=0;i<c;i++)
{
for(int j=0;j<r;j++)
{
plainText += mat[j][i];
}
}
return plainText;
}
}
```

Output:



❖ Simple Columnar Technique
Code:

```
package simplecolumnar;
import java.io.*;
public class SimpleColumnar

{
static String s1, st, d;
static StringBuffer s;
static int m, n, c, choice, p, q, k;
static int z[] = new int[10];
static char a[][];
public static void dis()
{
System.out.println();
System.out.println("Matrix:");
for(int i=0;i<m;i++)
{
for(int j=0;j<n;j++)
{
if(a[i][j]!='$')
System.out.print(a[i][j]+" ");
else
System.out.print(" ");
}
System.out.println();
}
System.out.println();
}
public static void enc(DataInputStream dis) throws Exception
{
while(true)
{
c = 0;
s1 = "";
System.out.println("-------------------------------------");
System.out.println();
System.out.print("Enter columns Sequence between 1 to "+n+":");
st = dis.readLine();
d = st+d;
for(int i=0;i<n;i++)
{
c = (int)st.charAt(i)-49;
for(int j=0;j<m;j++)
{
if(a[j][c]!='$')
s1=s1+a[j][c];
```

```
}
}
s1.trim();
c = 0;
for(int i=0;i<m;i++)
for(int j=0;j<n;j++)
if(c<s1.length())

a[i][j]=s1.charAt(c++);
else
a[i][j]='$';
dis();
System.out.print("Do You want to continue(yes(1)/no(0)):");
choice = Integer.parseInt(dis.readLine());
if(choice == 0)
{
System.out.println("--------------------------------------------------");
System.out.println();
System.out.println("Ecryption results in the ciphertext:"+s1);
return;
}
}
}
public static void dec()
{
k = 0;
p = s1.length()/n;
q = s1.length()%n;
for(int i=0;i<m;i++)
for(int j=0;j<n;j++)
a[i][j]='$';
for(int i=0;i<d.length();i++)
{
c = (int)d.charAt(i)-49;
if(c >= q)
{
for(int j=0;j<p;j++)
{
a[j][c]=s1.charAt(k++);
}
}
else
{
for(int j=0;j<p+1;j++)
{
a[j][c]=s1.charAt(k++);
}
}
dis();
if(k == s1.length())
```

```
{
s1 = "";
k = 0;
for(int x=0;x<m;x++)

for(int j=0;j<n;j++)
if(a[x][j] != '$')
{
s1 = s1+a[x][j];
a[x][j] = '$';
}
}
}
System.out.println("Decryption results in the plaintext:"+s1);
}
public static void main(String[] args)
{
try
{
DataInputStream dis = new DataInputStream(System.in);
System.out.print("Enter Plain text:");
s1 = dis.readLine();
s = new StringBuffer(s1);
//REMOVING WIDE-SPACES
for(int i=0;i<s.length();i++)
if(s.charAt(i) ==' ')
s.deleteCharAt(i);
s1 = new String(s);
d = "";
System.out.println("Enter size of the array:");
System.out.print("Enter number of rows: ");
m = Integer.parseInt(dis.readLine());
System.out.print("Enter no of columns:");
n = Integer.parseInt(dis.readLine());
a = new char[m][n];
c = 0;
//ENTERING IN THE ARRAY
for(int i=0;i<m;i++)
for(int j=0;j<n;j++)
if(c<s1.length())
a[i][j] = s1.charAt(c++);
else
a[i][j]='$';
dis();
enc(dis);
System.out.println();
System.out.println("----------------Decryption-----------------");
dec();
}
catch(Exception e)
```

```
{}

}
}
```

Output:

```
Output - SimpleColumnar (run) ×
Output - SimpleColumnar (run)
Enter Plain text:pizzalike
Enter size of the array:
Enter number of rows: 3
Enter no of columns:3

Matrix:
p i z
z a l
i k e

------------------------------------

Enter columns Sequence between 1 to 3:312

Matrix:
z l e
p z i
i a k

Do You want to continue(yes(1)/no(0)):0
---------------------------------------------------

Ecryption results in the ciphertext:zlepziiak
```

```
----------------Decryption-----------------

Matrix:
  z
  l
  e


Matrix:
p  z
z  l
i  e


Matrix:
p i z
z a l
i k e

Decryption results in the plaintext:pizzalike
BUILD SUCCESSFUL (total time: 59 seconds)
```

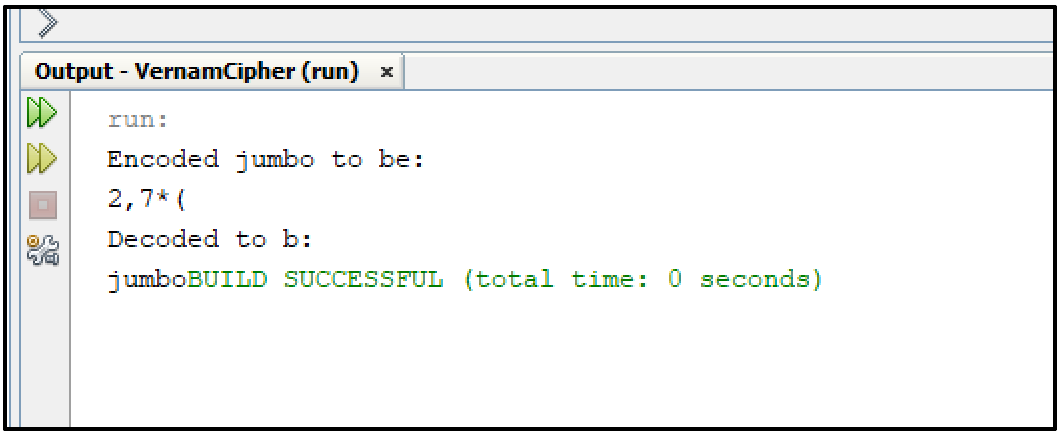
❖ Vermin Cipher

Code:

```
package vernamcipher;
import java.lang.Math;
public class VernamCipher {
public static void main(String args[])
{
String text = new String("jumbo");

char[] arText = text.toCharArray();
String cipher = new String("XYZHG");
char[] arCipher = cipher.toCharArray();
char[] encoded = new char[5];
System.out.println("Encoded " + text + " to be:");
for (int i = 0; i < arText.length; i++)
{
encoded[i] = (char) (arText[i] ^ arCipher[i]);
System.out.print(encoded[i]);
}
System.out.println();
System.out.println("Decoded to b:");
for (int i = 0; i < encoded.length; i++)
{
char temp = (char) (encoded[i] ^ arCipher[i]);
System.out.print(temp);
}
}
}
}
```

Output:



Conclusion:

Implementation of different types of cipher for Encryption and Decryption is successfully executed

Practical No.:3

Aim: Write a program to implement the (i) RSA Algorithm to perform encryption and decryption

Theory:

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography:

1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client's public key and sends the encrypted data.
3. The client receives this data and decrypts it.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is a multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task.

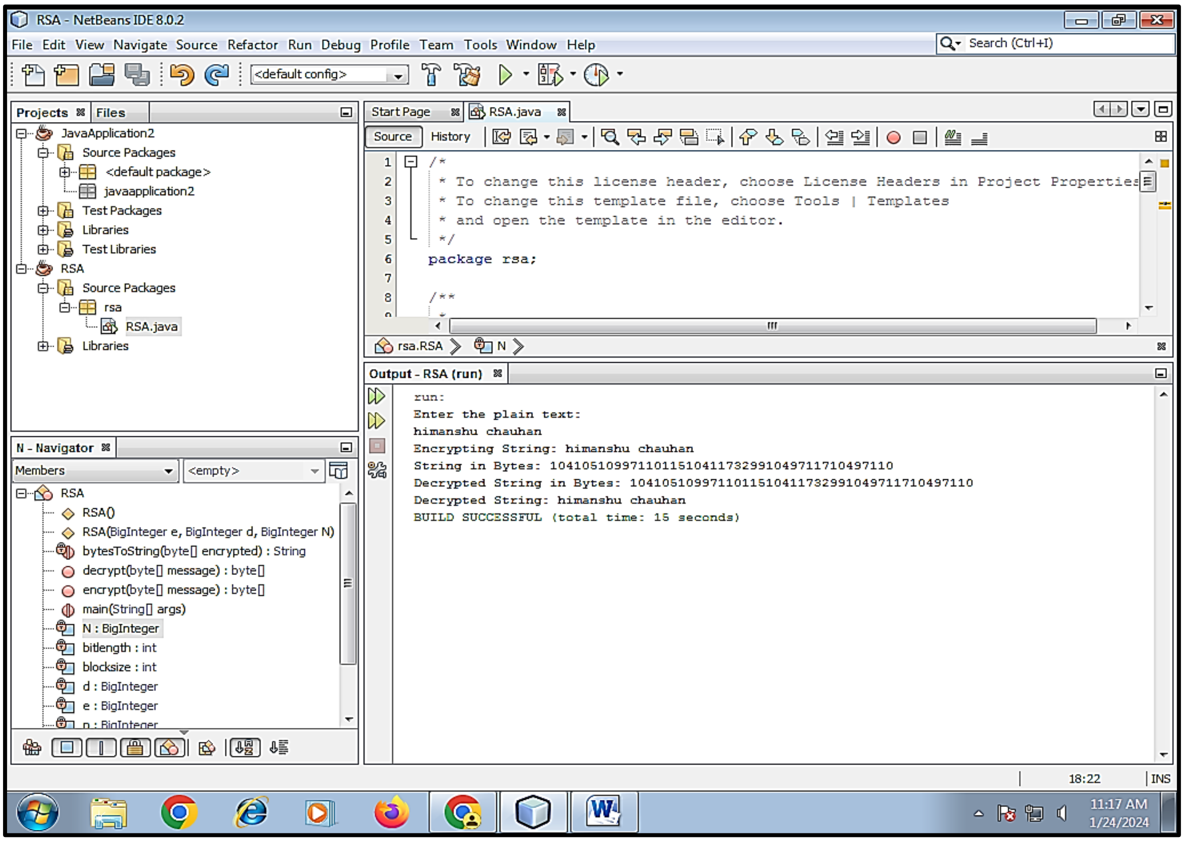
Code:

```
package rsa;
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
public class RSA {
private BigInteger p;
private BigInteger q;
private BigInteger N;
private BigInteger phi;
private BigInteger e;
private BigInteger d;
private int bitlength = 1024;
private int blocksize = 256; //blocksize in byte
private Random r;
public RSA() {
r = new Random();
p = BigInteger.probablePrime(bitlength, r);
q = BigInteger.probablePrime(bitlength, r);
N = p.multiply(q);
phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e = BigInteger.probablePrime(bitlength/2, r);
```

```
while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0 ) {
e.add(BigInteger.ONE);
}
d = e.modInverse(phi);
}
public RSA(BigInteger e, BigInteger d, BigInteger N) {
this.e = e;
this.d = d;
this.N = N;
}
public static void main (String[] args) throws IOException
{

RSA rsa = new RSA();
DataInputStream in=new DataInputStream(System.in);
String teststring ;
System.out.println("Enter the plain text:");
teststring=in.readLine();
System.out.println("Encrypting String: " + teststring);
System.out.println("String in Bytes: " + bytesToString(teststring.getBytes()));
// encrypt
byte[] encrypted = rsa.encrypt(teststring.getBytes());
// decrypt
byte[] decrypted = rsa.decrypt(encrypted);
System.out.println("Decrypted String in Bytes: " + bytesToString(decrypted));
System.out.println("Decrypted String: " + new String(decrypted));
}
private static String bytesToString(byte[] encrypted) {
String test = "";
for (byte b : encrypted) {
test += Byte.toString(b);
}
return test;
}
//Encrypt message
public byte[] encrypt(byte[] message) {
return (new BigInteger(message)).modPow(e, N).toByteArray();
}
// Decrypt message
public byte[] decrypt(byte[] message) {
return (new BigInteger(message)).modPow(d, N).toByteArray();
}
}
```

Output:



Conclusion:

RSA algorithm is implemented with the given string which gives bytes in the program.

Practical No.: 4

Aim:Write a program to implement the
(i)Miller-Rabin Algorithm
(ii) pollard p-1 Algorithm to perform encryption and decryption

Theory:

Miller Rabin:

Given a number n, check if it is prime or not. We have introduced and discussed School and Fermat methods for primality testing.
In this post, the Miller-Rabin method is discussed. This method is a probabilistic method (like Fermat), but it is generally preferred over Fermat’s method.

Algorithm:

// It returns false if n is composite and returns true if n

// is probably prime. k is an input parameter that determines

// accuracy level. Higher value of k indicates more accuracy.

bool isPrime(int n, int k)
1) Handle base cases for n < 3
2) If n is even, return false.
3) Find an odd number d such that n-1 can be written as $d*2^r$.
   Note that since n is odd, (n-1) must be even and r must be
   greater than 0.
4) Do following k times
     if (millerTest(n, d) == false)
          return false
5) Return true.

// This function is called for all k trials. It returns
// false if n is composite and returns true if n is probably
// prime.
// d is an odd number such that $d*2^r$ = n-1 for some r>=1
bool millerTest(int n, int d)
1) Pick a random number 'a' in range [2, n-2]
2) Compute: x = pow(a, d) % n
3) If x == 1 or x == n-1, return true.

// Below loop mainly runs 'r-1' times.
4) Do following while d doesn't become n-1.
     a) x = (x*x) % n.
     b) If (x == 1) return false.
     c) If (x == n-1) return true.

<u>Pollard P1:-</u>

Factorizing a large odd integer, **n**, into its corresponding prime factors can prove to be a difficult task. A brute approach can be testing all integers less than n until a divisor is found. This proves to be very time consuming as a divisor might be a very large prime itself. **Pollard p-1 algorithm** is a better approach to find out prime factors of any integer.

Using the combined help of Modular Exponentiation and GCD, it is able to calculate all the distinct prime factors in no time.

Algorithm

- Given a number n. Initialize a = 2, i = 2
- Until a factor is returned do a <- (a^i) mod n d <- GCD(a-1, n) if 1 < d < n then return d else i <- i+1
- Other factor, d’ <- n/d
- If d’ is not prime n <- d’ go to 1 else d and d’ are two prime factors

Code:

- Miler Rabin Algortihm

```
// Java program Miller-Rabin primality test
import java.io.*;
import java.math.*;
class GFG {
        // Utility function to do modular
        // exponentiation. It returns (x^y) % p
        static int power(int x, int y, int p) {
                int res = 1; // Initialize result
                //Update x if it is more than or
                // equal to p
                x = x % p;
                while (y > 0) {
                        // If y is odd, multiply x with result
                        if ((y & 1) == 1)
                                res = (res * x) % p;

                        // y must be even now
                        y = y >> 1; // y = y/2
                        x = (x * x) % p;
                }

                return res;
        }
        static boolean miillerTest(int d, int n) {

                // Pick a random number in [2..n-2]
                // Corner cases make sure that n > 4
                int a = 2 + (int)(Math.random() % (n - 4));
                // Compute a^d % n
                int x = power(a, d, n);
                if (x == 1 || x == n - 1)
                        return true;
                // Keep squaring x while one of the
                // following doesn't happen
                // (i) d does not reach n-1
                // (ii) (x^2) % n is not 1
                // (iii) (x^2) % n is not n-1
                while (d != n - 1) {
```

```
                x = (x * x) % n;
                d *= 2;

                if (x == 1)
                    return false;
                if (x == n - 1)
                    return true;
            }
            // Return composite
            return false;
        }
.
        static boolean isPrime(int n, int k) {

            // Corner cases
            if (n <= 1 || n == 4)
                return false;
            if (n <= 3)
                return true;

            // Find r such that n = 2^d * r + 1
            // for some r >= 1
            int d = n - 1;

            while (d % 2 == 0)
                d /= 2;

            // Iterate given number of 'k' times
            for (int i = 0; i < k; i++)
                if (!miillerTest(d, n))
                    return false;

            return true;
        }

        // Driver program
        public static void main(String args[]) {

            int k = 4; // Number of iterations

            System.out.println("All primes smaller "
                                            + "than 100: ");

            for (int n = 1; n < 100; n++)
                if (isPrime(n, k))
                    System.out.print(n + " ");
        }
}
```

Output:

```
All primes smaller than 100:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

❖ Pollard P1

Code:

```
// Java code for Pollard p-1
// factorization Method
import java.util.*;
class GFG
{
// function for
// calculating GCD
static long gcd(long a, long b)
{
        if (a == 0)
        return b;
        return gcd(b % a, a);
}

// function for
// checking prime
static boolean isPrime(long n)
{
        if (n <= 1)
        return false;
        if (n == 2 || n == 3)
        return true;
        if (n % 2 == 0)
        return false;
        for (long i = 3; i * i <= n; i += 2)
        if (n % i == 0)
                return false;
        return true;
}
// function to generate
// prime factors
static long pollard(long n)
{
        // defining base
        long a = 2;
        // defining exponent
        long i = 2;
```

```
            // iterate till a prime factor is obtained
            while(true)
            {
            // recomputing a as required
            a = ((long) Math.pow(a, i)) % n;
            a += n;
            a %= n;
            // finding gcd of a-1 and n
            // using math function
            long d = gcd(a-1,n);

            // check if factor obtained
            if (d > 1)
            {
                    //return the factor
                    return d;
            }

            // else increase exponent by one
            // for next round
            i += 1;
            }
    }

    // Driver code
    public static void main(String[] args)
    {
            long n = 1403;
            // temporarily storing n
            long num = n;
            // list for storing prime factors
            ArrayList<Long> ans = new ArrayList<Long>();

            // iterated till all prime factors
            // are obtained
            while(true)
            {
            // function call
            long d = pollard(num);

            // add obtained factor to list
            ans.add(d);

            // reduce n
            long r = (num/d);

            // check for prime
            if(isPrime(r))
            {
                    // both prime factors obtained
```

```
                ans.add(r);
                break;
            }
            // reduced n is not prime, so repeat
            else
                num = r;
            }
            // prlong the result
            System.out.print("Prime factors of " + n + " are ");
            for (long elem : ans)
            System.out.print(elem + " ");
}
}
```

Output:

```
Prime factors of 1403 are 61 23
```

Conclusion :

The Miller-rabin algorithm and Pollard p1 algorithm doesn’t use for encryption and decryption method ,Miller rabin used for giving prime number at certain range and Pollard p1 for giving prime factors of a number.

Practical No.:5

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

Theory:

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.

- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

  - Step 1: Alice and Bob get public numbers P = 23, G = 9

    Step 2: Alice selected a private key a = 4 and
    Bob selected a private key b = 3

    Step 3: Alice and Bob compute public values
    Alice: x =(9^4 mod 23) = (6561 mod 23) = 6
    Bob: y = (9^3 mod 23) = (729 mod 23) = 16

    Step 4: Alice and Bob exchange public numbers

    Step 5: Alice receives public key y =16 and
    Bob receives public key x = 6

    Step 6: Alice and Bob compute symmetric keys
    Alice: ka = y^a mod p = 65536 mod 23 = 9
    Bob: kb = x^b mod p = 216 mod 23 = 9

    Step 7: 9 is the shared secret.

Code :

```
package diffie.hellman;

import java.security.*;
import java.security.spec.*;
import java.io.*;
public class DiffieHellman
{
public static void generateKey(String keyAlgorithm, int numBits)
```

```
{
try
{
KeyPairGenerator keyGen = KeyPairGenerator.getInstance(keyAlgorithm);
keyGen.initialize(numBits);
KeyPair keyPair = keyGen.genKeyPair();
PrivateKey privateKey = keyPair.getPrivate();
PublicKey publicKey = keyPair.getPublic();
System.out.println("\n" + "Generating Key/value pair using " +
privateKey.getAlgorithm() + " algorithm");
byte privateKeyBytes[] = privateKey.getEncoded();
byte publicKeyBytes[] = publicKey.getEncoded();
String formatPrivate = privateKey.getFormat();
String formatPublic = publicKey.getFormat();
System.out.println("Private key format : " + formatPrivate);
System.out.println("Public key format : " + formatPublic);
KeyFactory keyFactory = KeyFactory.getInstance(keyAlgorithm);
EncodedKeySpec privateKeySpec = new
PKCS8EncodedKeySpec(privateKeyBytes);
PrivateKey privateKey1 = keyFactory.generatePrivate(privateKeySpec);
EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(publicKeyBytes);
PublicKey publicKey1 = keyFactory.generatePublic(publicKeySpec);
System.out.println("Are both private keys equal?" +
privateKey.equals(privateKey1));
System.out.println("Are both public keys equal?" +
publicKey.equals(publicKey1));

}
catch(InvalidKeySpecException e)
{
System.out.println(e);
}
catch(NoSuchAlgorithmException e)
{
System.out.println(e);
}
}
public static void main(String a[]) throws IOException
{
generateKey("DH", 576);
}
}
```

Output:

```
diffie.hellman.DiffieHellman
Output - Diffie-hellman (run)
run:

Generating Key/value pair using DH algorithm
Private key format : PKCS#8
Public key format : X.509
Are both private keys equal?true
Are both public keys equal?true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusion:
The implementation of Diffie-Hellman Algorithm is executed using public and private key and checks both are true or not.

Practical No.:6

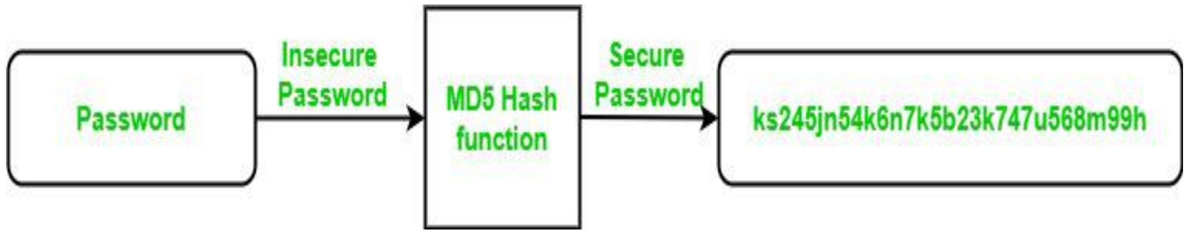
Aim: Write a program to implement the MD5 algorithm compute the message digest**.**

Theory:

MD5 is a cryptographic hash function algorithm that takes the message as input of any length and changes it into a fixed-length message of 16 bytes. MD5 algorithm stands for the **message-digest algorithm**. MD5 was developed as an improvement of MD4, with advanced security purposes. The output of MD5 (Digest size) is always **128 bits. MD5** was developed in 1991 by **Ronald Rivest.**

Use Of MD5 Algorithm:

- It is used for file authentication.
- In a web application, it is used for security purposes. e.g. Secure password of users etc.
- Using this algorithm, We can store our password in 128 bits format.



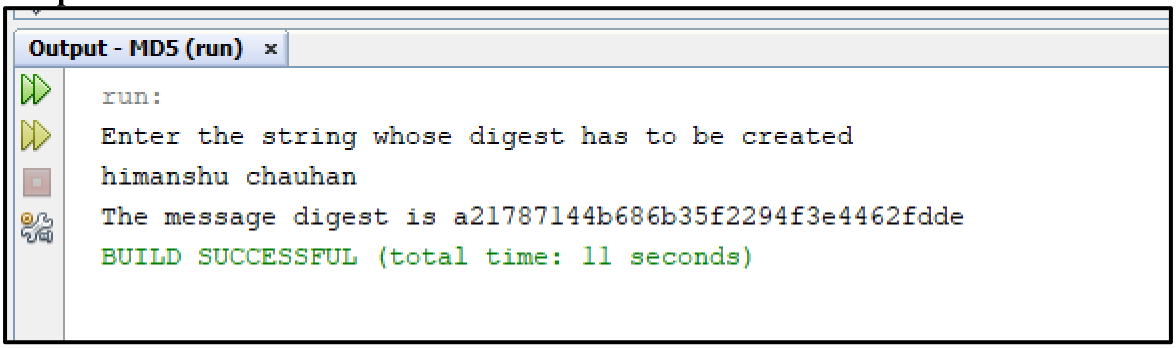
MD5 Algorithm

Code:

```
package md5;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;
public class MD5
{
public static void main(String[] args)
{
System.out.println("Enter the string whose digest has to be created");
Scanner s=new Scanner(System.in);
String str=s.nextLine();
System.out.println("The message digest is "+makeDigest(str));
}
public static String makeDigest(String input)
{
String md = null;
try
{
//Create MessageDigest object for MD5
MessageDigest digest = MessageDigest.getInstance("MD5");
//Update input string in message digest
digest.update(input.getBytes(), 0, input.length());
```

```
//Converts message digest value in base 16 (hex)
md= new BigInteger(1, digest.digest()).toString(16);
}
catch (NoSuchAlgorithmException e)
{
e.printStackTrace();
}
return md;
}
}
```

Output:



Conclusion:
MD5 algorithm is used to hash the string given by the user which is successfully implemented.

Practical No.:7

Aim: Write a program to implement HMAC signatures

Theory:

HMAC (Hash-based Message Authentication Code) is a type of a message authentication code (MAC) that is acquired by executing a cryptographic hash function on the data (that is) to be authenticated and a secret shared key. Like any of the MAC, it is used for both data integrity and authentication. Checking data integrity is necessary for the parties involved in communication. HTTPS, SFTP, FTPS, and other transfer protocols use HMAC. The cryptographic hash function may be MD-5, SHA-1, or SHA-256. Digital signatures are nearly similar to HMACs i.e they both employ a hash function and a shared key. The difference lies in the keys i.e HMACs use symmetric key(same copy) while Signatures use asymmetric (two different keys)

The formula **for HMAC:**
HMAC = hashFunc(secret key + message)

Code:

```
package hmac;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.security.SecureRandom;
import sun.misc.*;
public class HMAC
{
public static void main (String[] args) throws Exception
{
SecureRandom random = new SecureRandom();
byte[] keyBytes = new byte[20];
random.nextBytes(keyBytes);
SecretKeySpec key = new SecretKeySpec(keyBytes, "HMACSHA1");
System.out.println("Key:"+new BASE64Encoder().encode(key.getEncoded()));
Mac mac = Mac.getInstance("HmacSHA1");
mac.init(key);
mac.update("hello".getBytes("UTF8"));
byte[] result = mac.doFinal();
System.out.println("MAC: "+new BASE64Encoder().encode(result));
}
}
```

Output:

Output - HMAC (run)

```
run:
Key:7Us+08kReMQytI3inl9InkXcuA0=
MAC: VZgvBFsdeuvROKh4S27+rZf4JQQ=
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusion:

HMAC technique used the secret key randomly and message authentication code which is implemented .