MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF CS(CYBER SECURITY)

CERTIFICATE

This is to certify that Ms./Mr. ……..…… Reg. No. …..…… Section: …… Roll No..... has satisfactorily completed the lab exercises prescribed for NUMBER THEORY AND CRYPTOGRAPHY LABORATORY [IT_3161] of Third Year B. Tech. Degree at MIT, Bengaluru, in the academic year 2024.

Date: …….....

Signature
Faculty in Charge

CONTENTS

Course Objectives

- Understand the cryptography, Understand the role of encryption algorithm.
- Understand to converting some secret information to not readable texts
- Understand practice of hiding information

Course Outcomes

At the end of this course, students will have the

- Identify information system requirements for both of them such as client and server
- Understand basic cryptographic algorithms, message and web authentication and security issues.
- Understand the current legal issues towards information security

Evaluation plan

- Internal Assessment Marks : 60 marks

  Continuous evaluation: 40 Marks

  - The Continuous evaluation assessment will depend on punctuality, designing right algorithm, converting algorithm into an efficient program, maintaining the observation note and answering the questions in viva voce.
  - Internal Exam :20 Marks

- End semester assessment of 2 hour duration: 40 marks

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
    - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs and outputs.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.
    - Make use of constants and type definitions wherever needed.
    - Programs should include necessary time analysis part (Operation count /Step count method)
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
    - Solved exercise

    - Lab exercises - to be completed during lab hours
    - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition lab with the permission of the faculty concerned.
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample note preparation is given as a model for observation.
- You may write scripts/programs to automate the experimental analysis of algorithms and compare with the theoretical result.
- You may use spreadsheets to plot the graph.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

Number Theory Algorithms-I

Objectives:

In this lab, student will be able to:

- Understand **Number Theory algorithms**
- Understand the relationship between variables

Description: Number theory is a branch of mathematics that deals with the properties and relationships of integers and their fundamental properties. It has numerous applications in various fields, including cryptography, computer science, and algorithms. When it comes to number theory algorithms, the main objectives are to efficiently solve problems and perform computations related to integers and their properties.

I. SOLVED EXERCISE:

1. Implement the Euclidean algorithm in Java to find the greatest common divisor (GCD) of two integers.

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

    // Method to compute the GCD of two integers using the Euclidean algorithm
    public static int gcd(int a, int b) {
        // Ensure the numbers are non-negative
        if (a < 0) a = -a;
        if (b < 0) b = -b;

        // Base case: if b is zero, gcd is a
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Main method to get user input and display the GCD
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the first integer: ");
        int num1 = scanner.nextInt();

        System.out.print("Enter the second integer: ");
        int num2 = scanner.nextInt();
```

```
        // Compute the GCD
        int result = gcd(num1, num2);

        // Display the result
        System.out.println("The GCD of " + num1 + " and " + num2 + " is: " + result);

        // Close the scanner
        scanner.close();
    }
}
```

II LAB EXERCISES

1) Write a program to compute GCD of a list of integers.

2) Write a Java program to compute the greatest common divisor (GCD) of two integers using a recursive implementation of the Euclidean algorithm. The program should prompt the user to input the two integers and then display their GCD.

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

LAB NO: 2 **Date:**

Number Theory Algorithms - II

Objectives:

In this lab, student will be able to:
- Understand Number Theory algorithms
- Understand the relationship between variables

Description: Number theory is a branch of mathematics that deals with the properties and relationships of integers and their fundamental properties. It has numerous applications in various fields, including cryptography, computer science, and algorithms. When it comes to number theory algorithms, the main objectives are to efficiently solve problems and perform computations related to integers and their properties.

I. SOLVED EXERCISE:

1. Java program to implement the Extended Euclidean Algorithm to compute the Greatest Common Divisor (GCD).

```
import java.util.Scanner;

public class ExtendedEuclideanAlgorithm {

    // Method to compute the GCD and the coefficients x and y
    public static int extendedGCD(int a, int b, int[] x, int[] y) {
        // Base case: if b is zero, the GCD is a, and x = 1, y = 0
        if (b == 0) {
            x[0] = 1;
            y[0] = 0;
            return a;
        }

        // Recursive case: a = b * q + r
        int[] x1 = new int[1];
        int[] y1 = new int[1];
        int gcd = extendedGCD(b, a % b, x1, y1);

        // Update x and y using the results of the recursive call
        x[0] = y1[0];
        y[0] = x1[0] - (a / b) * y1[0];

        return gcd;
    }

    // Main method to get user input and display the GCD and coefficients
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the first integer (a): ");
        int a = scanner.nextInt();

        System.out.print("Enter the second integer (b): ");
```

```
        int b = scanner.nextInt();

        int[] x = new int[1];
        int[] y = new int[1];

        // Compute the GCD and the coefficients
        int gcd = extendedGCD(a, b, x, y);

        // Display the results
        System.out.println("The GCD of " + a + " and " + b + " is: " + gcd);
        System.out.println("Coefficients: x = " + x[0] + ", y = " + y[0]);

        // Close the scanner
        scanner.close();
    }
}
```

Lab exercises

1. Write a Java program that uses the Extended Euclidean Algorithm to solve the linear Diophantine equation ax+by=cax + by = cax+by=c. The program should prompt the user for integers aaa, bbb, and ccc, and then compute and display a particular solution (x,y)(x, y)(x,y) if one exists. If no solution exists, the program should indicate this.
2. Write a Java program that extends the Extended Euclidean Algorithm to find all integer solutions of the linear Diophantine equation ax+by=cax + by = cax+by=c. The program should prompt the user for integers aaa, bbb, and ccc, as well as a range value kkk. The program should compute and display a particular solution and then generate and display the general solutions (x,y)(x, y)(x,y) for different values of kkk within the specified range

[OBSERVATION SPACE – LAB2]

[OBSERVATION SPACE – LAB2]

LAB NO: 3 **Date:**

Number Theory Algorithms-III

Objectives:

In this lab, student will be able to:

- Understand Number Theory algorithms
- Understand the relationship between variables

Description: Number theory is a branch of mathematics that deals with the properties and relationships of integers and their fundamental properties. It has numerous applications in various fields, including cryptography, computer science, and algorithms. When it comes to number theory algorithms, the main objectives are to efficiently solve problems and perform computations related to integers and their properties.

I. SOLVED EXERCISE:

1) Implement the Chinese Remainder Theorem algorithm in Java to solve a system of linear congruences.

Description: According to the Chinese Remainder Theorem in Mathematics, if one is aware of the remainders of the Euclidean division of an integer n by several integers, they can then be used to determine the unique remainder of n's division by the product of these other integers, provided that the n and the divisors are pairwise coprime (no two divisors share a common factor other than 1).

Program

```
import java.util.Scanner;

public class ChineseRemainderTheorem {

   // Extended Euclidean Algorithm to find the modular inverse of 'a' modulo 'm'
   private static long modInverse(long a, long m) {
      long m0 = m, t, q;
      long x0 = 0, x1 = 1;
   if (m == 1)
    return 0;

   while (a > 1) {
         q = a / m;
         t = m;
         m = a % m;
         a = t;
         t = x0;
         x0 = x1 - q * x0;
         x1 = t;
      }
      if (x1 < 0)
         x1 += m0;

      return x1;
   }
   public static long chineseRemainder(long[] num, long[] rem) {
      if (num.length != rem.length) {
         throw new IllegalArgumentException("Number of elements in num[] and rem[] must be
the same.");
      }
      int n = num.length;
      long product = 1;
      for (long value : num) {
         product *= value;
      }

      long result = 0;
      for (int i = 0; i < n; i++) {
         long pp = product / num[i];
         result += rem[i] * modInverse(pp, num[i]) * pp;
         result %= product;
      }

      return result;
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of congruences: ");
      int n = scanner.nextInt();

      long[] num = new long[n];
      long[] rem = new long[n];

      System.out.println("Enter the values for a and m for each congruence:");
```

```
        for (int i = 0; i < n; i++) {
            System.out.print("a" + (i + 1) + ": ");
            num[i] = scanner.nextLong();
            System.out.print("m" + (i + 1) + ": ");
            rem[i] = scanner.nextLong();
        }

        long result = chineseRemainder(num, rem);
        System.out.println("The solution for the system of linear congruences is: " + result);
    }
}
```

Lab Exercises

Question 1: Solve a System of Linear Congruences

Write a Java program to implement the Chinese Remainder Theorem (CRT) to solve a system of linear congruences. The program should prompt the user to input a set of congruences of the form $x \equiv a_i \pmod{m_i}$, where a_i and m_i are given integers. The program should compute and display the solution x that satisfies all the given congruences, or indicate if no such solution exists.

Question 2: Find the Smallest Non-Negative Solution

Develop a Java program that uses the Chinese Remainder Theorem (CRT) to find the smallest non-negative integer solution to a system of linear congruences. The user should input multiple pairs of integers (a_i, m_i), where each pair represents a congruence $x \equiv a_i \pmod{m_i}$. The program should compute and print the smallest non-negative integer x that satisfies all the congruences simultaneously. If no solution exists, the program should inform the user.

LAB NO: 4 **Date:**

Number Theory Algorithms-IV

Objectives:

In this lab, student will be able to:

- Understand Number Theory algorithms
- Understand the relationship between variables

Description: Number theory is a branch of mathematics that deals with the properties and relationships of integers and their fundamental properties. It has numerous applications in various fields, including cryptography, computer science, and algorithms. When it comes to number theory algorithms, the main objectives are to efficiently solve problems and perform computations related to integers and their properties.

II. SOLVED EXERCISE:

1) Write a java program to implement Fermat's little theorem.

Fermat's Little Theorem is a fundamental principle in number theory that provides a relationship between prime numbers and modular arithmetic. Specifically, it states that if p is a prime number and a is any integer not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$. This means that when a is raised to the power of $p-1$ and divided by p, the remainder is always 1. This theorem is particularly useful in areas such as cryptography and primality testing, as it underpins many algorithms and concepts in modular arithmetic and number theory. The theorem is named after Pierre de Fermat, a French mathematician, who first conjectured it in the 17th century.

```
import java.util.Scanner;

public class FermatsLittleTheorem {

   // Method to compute a^b % mod using modular exponentiation
   public static long modularExponentiation(long base, long exponent, long mod) {
      long result = 1;
      base = base % mod; // Handle base greater than mod

      while (exponent > 0) {
         // If exponent is odd, multiply base with result
         if ((exponent % 2) == 1) {
            result = (result * base) % mod;
         }

         // Exponent must be even now
         exponent = exponent >> 1; // exponent = exponent / 2
         base = (base * base) % mod; // Change base to base^2
      }

      return result;
   }

   // Method to check Fermat's Little Theorem
   public static boolean verifyFermatsLittleTheorem(long a, long p) {
      // p should be a prime number
      if (p <= 1 || !isPrime(p)) {
         throw new IllegalArgumentException("p must be a prime number greater than
1.");
```

```
        }

        // Fermat's Little Theorem: a^(p-1) % p should be 1
        long result = modularExponentiation(a, p - 1, p);
        return result == 1;
    }

    // Method to check if a number is prime
    public static boolean isPrime(long number) {
        if (number <= 1) return false;
        if (number <= 3) return true;
        if (number % 2 == 0 || number % 3 == 0) return false;
        for (long i = 5; i * i <= number; i += 6) {
            if (number % i == 0 || number % (i + 2) == 0) return false;
        }
        return true;
    }

    // Main method to get user input and verify Fermat's Little Theorem
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the integer a: ");
        long a = scanner.nextLong();

        System.out.print("Enter the prime number p: ");
        long p = scanner.nextLong();

        try {
            boolean result = verifyFermatsLittleTheorem(a, p);
            System.out.println("Fermat's Little Theorem holds: " + result);
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }
}
```

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

LAB NO: 5 **Date:**

Algebra Properties

Objectives:

In this lab, student will be able to:

- Properties of Groups
- Properties of Ring

Description: A group is an algebraic structure consisting of a set of elements and a binary operation that satisfies four properties: closure, associativity, identity element, and invertibility. Closure means the operation on any two elements yields another element within the set. Associativity ensures the operation's grouping does not affect the result. The identity element leaves other elements unchanged when used in the operation, and invertibility means every element has an inverse that combines with it to produce the identity element.

A ring extends a group with an additional binary operation, typically addition and multiplication, satisfying extra properties. It requires closure and associativity for both operations, an additive identity, and additive inverses for each element. Moreover, the multiplication must distribute over addition, meaning multiplying a sum of elements yields the same result as summing their individual products. Rings provide a foundational framework in algebra, linking groups to fields and enabling various mathematical applications..

SOLVED EXERCISE:

1) Implement a Java class to represent a mathematical group and another to represent a ring, ensuring they satisfy the respective algebraic properties.

Group Class

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.Set;

public class Group {
   private Set<Integer> elements;

   // Constructor to initialize the group with a set of elements
   public Group(Set<Integer> elements) {
      this.elements = elements;
   }

   // Check if an element is in the group
   private boolean contains(int element) {
      return elements.contains(element);
   }

   // Closure property
   public boolean isClosedUnderOperation() {
      for (int a : elements) {
         for (int b : elements) {
            if (!contains(a + b)) {
               return false;
            }
         }
```

```
    }
    return true;
}

// Associativity property
public boolean isAssociative() {
    for (int a : elements) {
        for (int b : elements) {
            for (int c : elements) {
                if ((a + (b + c)) != ((a + b) + c)) {
                    return false;
                }
            }
        }
    }
    return true;
}

// Identity element
public Integer getIdentityElement() {
    for (int e : elements) {
        boolean isIdentity = true;
        for (int a : elements) {
            if ((e + a) != a || (a + e) != a) {
                isIdentity = false;
                break;
            }
        }
        if (isIdentity) {
            return e;
        }
    }
    return null;
}

// Inverse element
public boolean hasInverseElement() {
    Integer identity = getIdentityElement();
    if (identity == null) {
        return false;
    }
    for (int a : elements) {
        boolean hasInverse = false;
        for (int b : elements) {
            if ((a + b) == identity) {
                hasInverse = true;
                break;
            }
        }
        if (!hasInverse) {
            return false;
        }
    }
    return true;
}

// Check if the structure is a group
```

```
    public boolean isGroup() {
        return isClosedUnderOperation() && isAssociative() && getIdentityElement() !=
null && hasInverseElement();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Set<Integer> elements = new HashSet<Integer>(); // Explicit type parameters

        System.out.println("Enter the elements of the group (type 'done' to finish):");

        while (scanner.hasNextInt()) {
            elements.add(scanner.nextInt());
        }

        // Clear the scanner's buffer
        scanner.next(); // To consume the 'done' or invalid input

        Group group = new Group(elements);

        if (group.isGroup()) {
            System.out.println("The set forms a group under addition.");
        } else {
            System.out.println("The set does not form a group under addition.");
        }

        scanner.close();
    }
}
```

Ring Class

```
import java.util.Scanner;
import java.util.HashSet;
import java.util.Set;

public class Ring {
    private Set<Integer> elements;

    // Constructor to initialize the ring with a set of elements
    public Ring(Set<Integer> elements) {
        this.elements = elements;
    }

    // Check if an element is in the ring
    private boolean contains(int element) {
        return elements.contains(element);
    }

    // Closure property for addition and multiplication
    public boolean isClosedUnderAddition() {
        for (int a : elements) {
            for (int b : elements) {
                if (!contains(a + b)) {
                    return false;
                }
            }
```

```
    }
    return true;
}

public boolean isClosedUnderMultiplication() {
    for (int a : elements) {
        for (int b : elements) {
            if (!contains(a * b)) {
                return false;
            }
        }
    }
    return true;
}

// Associativity property for addition and multiplication
public boolean isAssociativeAddition() {
    for (int a : elements) {
        for (int b : elements) {
            for (int c : elements) {
                if ((a + (b + c)) != ((a + b) + c)) {
                    return false;
                }
            }
        }
    }
    return true;
}

public boolean isAssociativeMultiplication() {
    for (int a : elements) {
        for (int b : elements) {
            for (int c : elements) {
                if ((a * (b * c)) != ((a * b) * c)) {
                    return false;
                }
            }
        }
    }
    return true;
}

// Distributive property
public boolean isDistributive() {
    for (int a : elements) {
        for (int b : elements) {
            for (int c : elements) {
                if ((a * (b + c)) != ((a * b) + (a * c))) {
                    return false;
                }
            }
        }
    }
    return true;
}

// Identity element for addition
```

```
    public Integer getAdditiveIdentity() {
        for (int e : elements) {
            boolean isIdentity = true;
            for (int a : elements) {
                if ((e + a) != a || (a + e) != a) {
                    isIdentity = false;
                    break;
                }
            }
            if (isIdentity) {
                return e;
            }
        }
        return null;
    }

    // Check if the structure is a ring
    public boolean isRing() {
        return isClosedUnderAddition() && isClosedUnderMultiplication() &&
               isAssociativeAddition() && isAssociativeMultiplication() &&
               isDistributive() && getAdditiveIdentity() != null;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Set<Integer> elements = new HashSet<Integer>(); // Explicit type parameters

        System.out.println("Enter the elements of the ring (type 'done' to finish):");

        while (scanner.hasNextInt()) {
            elements.add(scanner.nextInt());
        }

        // Clear the scanner's buffer
        scanner.next(); // To consume the 'done' or invalid input

        Ring ring = new Ring(elements);

        if (ring.isRing()) {
            System.out.println("The set forms a ring.");
        } else {
            System.out.println("The set does not form a ring.");
        }

        scanner.close();
    }
}
```

LAB NO: 6 **Date:**

Symmetric Conventional Cryptographic Techniques

Objectives:

In this lab, student will be able to:

- recall the concepts learnt in Cryptography
- implement basic techniques

Description: A cryptosystem is an implementation of cryptographic techniques and their accompanying infrastructure to provide information security services. A cryptosystem is also referred to as a cipher system.

Types of Cryptosystems

Fundamentally, there are two types of cryptosystems based on the manner in which encryption-decryption is carried out in the system −Symmetric Key Encryption-Asymmetric Key Encryption. The main difference between these cryptosystems is the relationship between the encryption and the decryption key. Logically, in any cryptosystem, both the keys are closely associated. It is practically impossible to decrypt the cipher text with the key that is unrelated to the encryption key.

SOLVED EXERCISE:

1) Write a Java program to implement the Caesar Cipher encryption technique.

Description : The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on.

```
import java.util.Scanner;

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

        System.out.print("Enter the text to encrypt: ");
        String plainText = scanner.nextLine();

        System.out.print("Enter the key (a number from 1 to 25): ");
        int key = scanner.nextInt();

        String encryptedText = encrypt(plainText, key);
        System.out.println("Encrypted text: " + encryptedText);

        scanner.close();
    }
    public static String encrypt(String plainText, int key) {
        StringBuilder cipherText = new StringBuilder();

        for (int i = 0; i < plainText.length(); i++) {
            char ch = plainText.charAt(i);

            if (Character.isLetter(ch)) {
                char shifted = (char) (((ch - 'a' + key) % 26) + 'a');
                cipherText.append(shifted);
            } else {
                cipherText.append(ch);
            }
        }
       return cipherText.toString();
    }
}
```

Lab exercises

1) Write a program to Modify the Caesar Cipher program to include decryption functionality.

2) Write a program to read the plaintext from an input file, encrypts it with a specified shift value, and writes the ciphertext to an output file.

LAB NO: 7 **Date:**

Symmetric classic cryptographic techniques

Objectives:

In this lab, student will be able to:

- recall the concepts learnt in Cryptography
- implement basic techniques

Description: A cryptosystem is an implementation of cryptographic techniques and their accompanying infrastructure to provide information security services. A cryptosystem is also referred to as a cipher system.

Types of Cryptosystems

Fundamentally, there are two types of cryptosystems based on the manner in which encryption-decryption is carried out in the system −Symmetric Key Encryption-Asymmetric Key Encryption. The main difference between these cryptosystems is the relationship between the encryption and the decryption key. Logically, in any cryptosystem, both the keys are closely associated. It is practically impossible to decrypt the cipher text with the key that is unrelated to the encryption key.

SOLVED EXERCISE:

Implement a brute-force attack program in Java to crack a Caesar Cipher-encrypted message.

Cracking a Caesar Cipher involves deciphering a message encrypted by shifting each letter by a fixed number of positions in the alphabet. To crack it, you systematically try each of the 25 possible shifts until the message becomes readable. This process can be simplified using frequency analysis, where you compare the frequency of letters in the ciphertext to typical letter frequencies in the target language, often leading to a faster identification of the correct shift.

Program

```
public class CaesarCipherCracker {
    public static void main(String[] args) {
        String ciphertext = "JGNNQ YQTNF"; // Example Ciphertext
        crackCaesarCipher(ciphertext);
    }

    public static void crackCaesarCipher(String ciphertext) {
        for (int shift = 0; shift < 26; shift++) {
            String decryptedText = decrypt(ciphertext, shift);
            System.out.println("Shift " + shift + ": " + decryptedText);
        }
    }

    public static String decrypt(String text, int shift) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (Character.isLetter(c)) {
                char base = Character.isUpperCase(c) ? 'A' : 'a';
                c = (char) ((c - base - shift + 26) % 26 + base);
            }
            result.append(c);
        }
```

LAB EXERCISES

1). Extend the substitution cipher program to generate a random substitution key for each encryption.

2). Write a Java program to decrypt a substitution cipher-encrypted message without knowing the substitution key.

[OBSERVATION SPACE – LAB7]

[OBSERVATION SPACE – LAB7]

LAB NO: 8 **Date:**

Traditional Cipher-I

Objectives:

In this lab, student will be able to:

- recall the concepts learnt in Cryptography
- implement basic techniques

Description: A cryptosystem is an implementation of cryptographic techniques and their accompanying infrastructure to provide information security services. A cryptosystem is also referred to as a cipher system.

SOLVED EXERCISE

Develop a Java program to implement a columnar transposition cipher encryption technique.

Description: In this program, the user is prompted to enter the text to be encrypted and the k In this program, the user is prompted to enter the text to be encrypted and the key, which is a string representing the column order. The encrypt method takes the plain text and column order as input and returns the encrypted text.

Program

```
import java.util.Arrays;
import java.util.Scanner;

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

        System.out.print("Enter the text to encrypt: ");
        String plainText = scanner.nextLine();

        System.out.print("Enter the key (a string representing the column order): ");
        String columnOrder = scanner.nextLine();

        String encryptedText = encrypt(plainText, columnOrder);
        System.out.println("Encrypted text: " + encryptedText);

        scanner.close();
    }
```

```
public static String encrypt(String plainText, String columnOrder)
{
      int rows = (int) Math.ceil((double) plainText.length() / columnOrder.length());
      char[][] transpositionMatrix = new char[rows][columnOrder.length()];

            int index = 0;
      for (int i = 0; i < rows; i++) {
         for (int j = 0; j < columnOrder.length(); j++) {
            if (index < plainText.length()) {
               transpositionMatrix[i][j] = plainText.charAt(index);
               index++;
            } else {
               transpositionMatrix[i][j] = 'X'; // Padding with 'X' for incomplete cells
            }
         }
      }

char[] sortedColumnOrder = columnOrder.toCharArray();
      Arrays.sort(sortedColumnOrder);

      // Build the encrypted text by reading the matrix column-wise based on sorted
column order
      StringBuilder encryptedText = new StringBuilder();
      for (char ch : sortedColumnOrder) {
         int columnIndex = columnOrder.indexOf(ch);
         for (int i = 0; i < rows; i++) {
            encryptedText.append(transpositionMatrix[i][columnIndex]);
         }
      }
      return encryptedText.toString();
   }
}
```

LAB EXERCISES

1). Write a program to Modify the transposition cipher program to include decryption functionality.
2). Implement a Java program to perform a known-plaintext attack on a transposition cipher.

[OBSERVATION SPACE – LAB8]

[OBSERVATION SPACE – LAB8]

LAB NO: 9 **Date:**

Traditional Ciphers-II

Objectives:

In this lab, student will be able to:

- Apply traditional cipher techniques for Critical Thinking and Problem-Solving
- Understand the advancements in modern cryptography
-

I. SOLVED EXERCISE:

1). Write a Java program to implement the Vigenère Cipher encryption and decryption techniques.

Description: The vigenere cipher is an algorithm that is used to encrypting and decrypting the text. The vigenere cipher is an algorithm of encrypting an alphabetic text that uses a series of interwoven caesar ciphers. It is based on a keyword's letters. It is an example of a polyalphabetic substitution cipher. This algorithm is easy to understand and implement.

Program

```
import java.util.Scanner;

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

        System.out.print("Enter the text to encrypt: ");
        String plainText = scanner.nextLine();

        System.out.print("Enter the encryption key: ");
        String key = scanner.nextLine();
        String encryptedText = encrypt(plainText, key);
        System.out.println("Encrypted text: " + encryptedText);

        String decryptedText = decrypt(encryptedText, key);
        System.out.println("Decrypted text: " + decryptedText);

        scanner.close();
    }
    public static String encrypt(String plainText, String key) {
        StringBuilder encryptedText = new StringBuilder();

        int keyLength = key.length();
        int plainTextLength = plainText.length();

       for (int i = 0; i < plainTextLength; i++) {
          char plainChar = plainText.charAt(i);
          char keyChar = key.charAt(i % keyLength);
```

```
if (Character.isLetter(plainChar)) {
                char encryptedChar = encryptChar(plainChar, keyChar);
                encryptedText.append(encryptedChar);
            } else {
                encryptedText.append(plainChar);
            }
        }

        return encryptedText.toString();
    }
    public static String decrypt(String encryptedText, String key) {
        StringBuilder decryptedText = new StringBuilder();

        int keyLength = key.length();
        int encryptedTextLength = encryptedText.length();

        for (int i = 0; i < encryptedTextLength; i++) {
            char encryptedChar = encryptedText.charAt(i);
            char keyChar = key.charAt(i % keyLength);
            if (Character.isLetter(encryptedChar)) {
                char decryptedChar = decryptChar(encryptedChar, keyChar);
                decryptedText.append(decryptedChar);
            } else {
                decryptedText.append(encryptedChar);
            }
        }

        return decryptedText.toString();
    }
public static char encryptChar(char plainChar, char keyChar) {
        plainChar = Character.toUpperCase(plainChar);
        keyChar = Character.toUpperCase(keyChar);

        int plainCharValue = plainChar - 'A';
        int keyCharValue = keyChar - 'A';

        int encryptedCharValue = (plainCharValue + keyCharValue) % 26;
        return (char) (encryptedCharValue + 'A');
    }

    public static char decryptChar(char encryptedChar, char keyChar) {
        encryptedChar = Character.toUpperCase(encryptedChar);
        keyChar = Character.toUpperCase(keyChar);

        int encryptedCharValue = encryptedChar - 'A';
        int keyCharValue = keyChar - 'A';
```

```
        return (char) (encryptedCharValue + 'A');
    }

    public static char decryptChar(char encryptedChar, char keyChar) {
        encryptedChar = Character.toUpperCase(encryptedChar);
        keyChar = Character.toUpperCase(keyChar);

        int encryptedCharValue = encryptedChar - 'A';
        int keyCharValue = keyChar - 'A';
        int decryptedCharValue = (encryptedCharValue - keyCharValue + 26) % 26;

        return (char) (decryptedCharValue + 'A');
    }
}
```

LAB EXERCISES

1). Write a program to Enhance the Vigenère Cipher program to handle both uppercase and lowercase letters.
2). Implement a Java program to crack a Vigenère Cipher-encrypted message using frequency analysis.

LAB NO: 10 **Date:**

Traditional Cipher-III

I. SOLVED EXERCISE:

1). Write a Java program to implement the Playfair cipher encryption and decryption.

Description: Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a digraph (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then J is replaced by I. It means treat I and J as the same letter, accordingly.

Program

```
import java.util.Scanner;

public class PlayfairCipher {
    private static final char PAD_CHAR = 'X';

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

        System.out.print("Enter the key: ");
        String key = scanner.nextLine();

        System.out.print("Enter the plaintext: ");
        String plaintext = scanner.nextLine();

        String encryptedText = encrypt(plaintext, key);
        System.out.println("Encrypted text: " + encryptedText);

        String decryptedText = decrypt(encryptedText, key);
        System.out.println("Decrypted text: " + decryptedText);
    }
```

```
public static String encrypt(String plaintext, String key) {
    char[][] matrix = generateMatrix(key);

    String formattedPlaintext = formatPlaintext(plaintext);

    StringBuilder encryptedText = new StringBuilder();
    for (int i = 0; i < formattedPlaintext.length(); i += 2) {
        char char1 = formattedPlaintext.charAt(i);
        char char2 = formattedPlaintext.charAt(i + 1);
        int[] char1Position = findPosition(matrix, char1);
        int[] char2Position = findPosition(matrix, char2);

        char encryptedChar1;
        char encryptedChar2;
        if (char1Position[0] == char2Position[0]) { // Same row
            encryptedChar1 = matrix[char1Position[0]][(char1Position[1] + 1) % 5];
            encryptedChar2 = matrix[char2Position[0]][(char2Position[1] + 1) % 5];
        } else if (char1Position[1] == char2Position[1]) { // Same column
            encryptedChar1 = matrix[(char1Position[0] + 1) % 5][char1Position[1]];
            encryptedChar2 = matrix[(char2Position[0] + 1) % 5][char2Position[1]];
        } else { // Rectangle
            encryptedChar1 = matrix[char1Position[0]][char2Position[1]];
            encryptedChar2 = matrix[char2Position[0]][char1Position[1]];
        }

        encryptedText.append(encryptedChar1).append(encryptedChar2);
    }

    return encryptedText.toString();
}

public static String decrypt(String ciphertext, String key) {
    char[][] matrix = generateMatrix(key);

    StringBuilder decryptedText = new StringBuilder();
    for (int i = 0; i < ciphertext.length(); i += 2) {
        char char1 = ciphertext.charAt(i);
        char char2 = ciphertext.charAt(i + 1);
        int[] char1Position = findPosition(matrix, char1);
        int[] char2Position = findPosition(matrix, char2);
        char decryptedChar1;
        char decryptedChar2;
        if (char1Position[0] == char2Position[0]) { // Same row
            decryptedChar1 = matrix[char1Position[0]][(char1Position[1] + 4) % 5];
            decryptedChar2 = matrix[char2Position[0]][(char2Position[1] + 4) % 5];
        } else if (char1Position[1] == char2Position[1]) { // Same column
            decryptedChar1 = matrix[(char1Position[0] + 4) % 5][char1Position[1]];
            decryptedChar2 = matrix[(char2Position[0] + 4) % 5][char2Position[1]];
        } else { // Rectangle
            decryptedChar1 = matrix[char1Position[0]][char2Position[1]];
            decryptedChar2 = matrix[char2Position[0]][char1Position[1]];
        }
```

```
decryptedText.append(decryptedChar1).append(decryptedChar2);
        }

        return decryptedText.toString().replace(String.valueOf(PAD_CHAR), "");
    }
    private static char[][] generateMatrix(String key) {
        char[][] matrix = new char[5][5];
        String keyWithoutDuplicates = removeDuplicates(key +
"ABCDEFGHIJKLMNOPQRSTUVWXYZ");

        int keyIndex = 0;
        int rowIndex = 0;
        int columnIndex = 0;

        while (keyIndex < keyWithoutDuplicates.length()) {
            char currentChar = keyWithoutDuplicates.charAt(keyIndex);
            matrix[rowIndex][columnIndex] = currentChar;
            columnIndex++;
            if (columnIndex == 5) {
                columnIndex = 0;
                rowIndex++;
            }

            keyIndex++;
        }

        return matrix;
    }
    private static String formatPlaintext(String plaintext) {
        StringBuilder formattedText = new StringBuilder();
        char previousChar = plaintext.charAt(0);
        formattedText.append(previousChar);

        for (int i = 1; i < plaintext.length(); i++) {
            char currentChar = plaintext.charAt(i);
            if (currentChar == previousChar) {
                formattedText.append(PAD_CHAR).append(currentChar);
            } else {
                formattedText.append(currentChar);
                previousChar = currentChar;
            }
        }
        if (formattedText.length() % 2 != 0) {
            formattedText.append(PAD_CHAR);
        }
        return formattedText.toString();
    }
```

```
private static int[] findPosition(char[][] matrix, char target) {
        int[] position = new int[2];
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                                            if (matrix[i][j] == target) {
                    position[0] = i;
                    position[1] = j;
                    return position;
                }
            }
        }
        return position;
    }
    private static String removeDuplicates(String input) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            char currentChar = input.charAt(i);
            if (result.indexOf(String.valueOf(currentChar)) == -1) {
                result.append(currentChar);
            }
        }
        return result.toString();
    }
}
```

LAB EXERCISES

1). Design a program to generate a random Playfair cipher key based on a given passphrase.
2) Develop a Playfair cipher solver that can decrypt Playfair-encrypted messages without knowing the key.

[OBSERVATION SPACE – LAB10]

LAB NO: 11 **Date:**

Traditional Ciphers -IV

I. SOLVED EXERCISE:

1). Implement a Java program to encrypt and decrypt messages using the Rail Fence cipher.

Description : In the rail fence cipher, the plaintext is written downwards diagonally on successive "rails" of an imaginary fence, then moving up when the bottom rail is reached, down again when the top rail is reached, and so on until the whole plaintext is written out. The ciphertext is then read off in rows.

Program

```
import java.util.Scanner;

public class PlayfairCipher {
   private static final char PAD_CHAR = 'X';

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

      System.out.print("Enter the key: ");
      String key = scanner.nextLine();

      System.out.print("Enter the plaintext: ");
      String plaintext = scanner.nextLine();
      String encryptedText = encrypt(plaintext, key);
      System.out.println("Encrypted text: " + encryptedText);

      String decryptedText = decrypt(encryptedText, key);
      System.out.println("Decrypted text: " + decryptedText);
   }
public static String encrypt(String plaintext, String key)
{
      char[][] matrix = generateMatrix(key);

      String formattedPlaintext = formatPlaintext(plaintext);

      StringBuilder encryptedText = new StringBuilder();
      for (int i = 0; i < formattedPlaintext.length(); i += 2) {
         char char1 = formattedPlaintext.charAt(i);
         char char2 = formattedPlaintext.charAt(i + 1);
         int[] char1Position = findPosition(matrix, char1);
```

```
            int[] char2Position = findPosition(matrix, char2);

            char encryptedChar1;
            char encryptedChar2;
            if (char1Position[0] == char2Position[0]) { // Same row
               encryptedChar1 = matrix[char1Position[0]][(char1Position[1] + 1) % 5];
               encryptedChar2 = matrix[char2Position[0]][(char2Position[1] + 1) % 5];
            } else if (char1Position[1] == char2Position[1]) { // Same column
               encryptedChar1 = matrix[(char1Position[0] + 1) % 5][char1Position[1]];
               encryptedChar2 = matrix[(char2Position[0] + 1) % 5][char2Position[1]];
            } else { // Rectangle
               encryptedChar1 = matrix[char1Position[0]][char2Position[1]];
               encryptedChar2 = matrix[char2Position[0]][char1Position[1]];
            }
            encryptedText.append(encryptedChar1).append(encryptedChar2);
         }

         return encryptedText.toString();
      }
      public static String decrypt(String ciphertext, String key) {
         char[][] matrix = generateMatrix(key);

         StringBuilder decryptedText = new StringBuilder();
         for (int i = 0; i < ciphertext.length(); i += 2) {
            char char1 = ciphertext.charAt(i);
            char char2 = ciphertext.charAt(i + 1);
            int[] char1Position = findPosition(matrix, char1);
            int[] char2Position = findPosition(matrix, char2);
char decryptedChar1;
            char decryptedChar2;
            if (char1Position[0] == char2Position[0]) { // Same row
               decryptedChar1 = matrix[char1Position[0]][(char1Position[1] + 4) % 5];
               decryptedChar2 = matrix[char2Position[0]][(char2Position[1] + 4) % 5];
            } else if (char1Position[1] == char2Position[1]) { // Same column
               decryptedChar1 = matrix[(char1Position[0] + 4) % 5][char1Position[1]];
               decryptedChar2 = matrix[(char2Position[0] + 4) % 5][char2Position[1]];
            } else { // Rectangle
               decryptedChar1 = matrix[char1Position[0]][char2Position[1]];
               decryptedChar2 = matrix[char2Position[0]][char1Position[1]];
            }
decryptedText.append(decryptedChar1).append(decryptedChar2);
         }

         return decryptedText.toString().replace(String.valueOf(PAD_CHAR), "");
      }
      private static char[][] generateMatrix(String key) {
         char[][] matrix = new char[5][5];
         String keyWithoutDuplicates = removeDuplicates(key +
"ABCDEFGHIJKLMNOPQRSTUVWXYZ");

         int keyIndex = 0;
         int rowIndex = 0;
         int columnIndex = 0;

         while (keyIndex < keyWithoutDuplicates.length()) {
            char currentChar = keyWithoutDuplicates.charAt(keyIndex);
            matrix[rowIndex][columnIndex] = currentChar;
            columnIndex++;
```

```
        boolean down = false;
        int row = 0, col = 0;

        for (int i = 0; i < message.length(); i++) {
            if (row == 0 || row == rails - 1) {
                down = !down;
            }
            fence[row][col++] = message.charAt(i);

            if (down) {
                row++;
            } else {
                row--;
            }
        }

StringBuilder encryptedMessage = new StringBuilder();
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < message.length(); j++) {
                if (fence[i][j] != '\n') {
                    encryptedMessage.append(fence[i][j]);
                }
            }
        }

        return encryptedMessage.toString();
    }
    public static String decrypt(String encryptedMessage, int rails) {
        char[][] fence = new char[rails][encryptedMessage.length()];
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < encryptedMessage.length(); j++) {
                fence[i][j] = '\n';
            }
        }

        boolean down = false;
        int row = 0, col = 0;

        for (int i = 0; i < encryptedMessage.length(); i++) {
            if (row == 0 || row == rails - 1) {
                down = !down;
            }

            fence[row][col++] = '*';
            if (down) {
                row++;
            } else {
                row--;
            }
        }
        int index = 0;
        for (int i = 0; i < rails; i++) {
            for (int j = 0; j < encryptedMessage.length(); j++) {
                if (fence[i][j] == '*' && index < encryptedMessage.length()) {
                    fence[i][j] = encryptedMessage.charAt(index++);
                }
            }
        }
```

```
StringBuilder decryptedMessage = new StringBuilder();
        row = 0;
        col = 0;
        for (int i = 0; i < encryptedMessage.length(); i++) {
            if (row == 0 || row == rails - 1) {
                down = !down;
            }

            if (fence[row][col] != '*') {
                decryptedMessage.append(fence[row][col++]);
            }
            if (down) {
                row++;
            } else {
                row--;
            }
        }

        return decryptedMessage.toString();
    }
public static void main(String[] args) {
        String message = "Hello, World!";
        int rails = 3;

        System.out.println("Original Message: " + message);

        String encryptedMessage = encrypt(message, rails);
        System.out.println("Encrypted Message: " + encryptedMessage);

        String decryptedMessage = decrypt(encryptedMessage, rails);
        System.out.println("Decrypted Message: " + decryptedMessage);
    }
 }
```

II. LAB EXERCISE:

Write a program to:

1) Enhance the Rail Fence cipher algorithm to support different fence heights.
2) Create a program to crack the Rail Fence cipher by trying different fence heights.

[OBSERVATION SPACE – LAB11]

[OBSERVATION SPACE – LAB11]

[OBSERVATION SPACE – LAB11]