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 [CSE_3121] of Third Year B. Tech. Degree at MIT, Bengaluru, in the academic year 2023.

Date: ……...................................

Signature
Faculty in Charge

CONTENTS

LAB NO.	TITLE	PAGE NO.	REMARKS
	COURSE OBJECTIVES AND OUTCOMES	3	
	EVALUATION PLAN	3	
	INSTRUCTIONS TO THE STUDENTS	4	
1	Symmetric conventional cryptographic techniques	6	
2	Symmetric classic cryptographic techniques	20	
3	Traditional Cipher-I	29	
4	Traditional Ciphers-II	38	
5	Traditional Ciphers-III	49	
6	Traditional Ciphers -IV	61	
7	Number Theory Algorithms	74	
8	Number Theory Algorithms -II	85	
9	Homomorphic encryption techniques	97	
10	Secure communication	106	

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:**

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.

I. 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();
    }
}
```

II LAB EXERCISES

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

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

[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:**

Symmetric Classic Cryptographic Techniques

Objectives:

In this lab, student will be able to:

- Familiarize with fundamentals of problem solving with the help of algorithms.
- Realize that for a problem there can be multiple solutions with different techniques.

Description: The Symmetric Classic Cryptographic Techniques lab is an educational and hands-on exercise designed to introduce participants to the fundamental concepts and practical implementation of symmetric encryption and decryption methods. The lab aims to familiarize learners with various classic cryptographic algorithms, which use a shared secret key for both encryption and decryption processes.

I. SOLVED EXERCISE:

1) Write a Java program that implements a simple substitution cipher, where each letter is replaced by a fixed substitution.

Description: The simple substitution cipher is a cipher that has been in use for many hundreds of years (an excellent history is given in Simon Singhs 'the Code Book'). It basically consists of substituting every plaintext character for a different cipher text character. It differs from the Caesar cipher in that the cipher alphabet is not simply the alphabet shifted, it is completely jumbled.

The simple substitution cipher offers very little communication security, and it will be shown that it can be easily broken even by hand, especially as the messages become longer (more than several hundred cipher text characters).

Program

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class SubstitutionCipher {
    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 substitution key: ");
        String substitutionKey = scanner.nextLine();

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

        scanner.close();
    }
 }
public static String encrypt(String plainText, String substitutionKey) {
        Map<Character, Character> substitutionMap =
generateSubstitutionMap(substitutionKey);

        StringBuilder cipherText = new StringBuilder();

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

            if (substitutionMap.containsKey(ch)) {
                char substituted = substitutionMap.get(ch);
                cipherText.append(Character.isLowerCase(plainText.charAt(i)) ? substituted :
Character.toUpperCase(substituted));
            } else {
                cipherText.append(ch);
            }
        }
        return cipherText.toString();
    }
```

```
public static Map<Character, Character> generateSubstitutionMap(String
substitutionKey)
{
        Map<Character, Character> substitutionMap = new HashMap<>();

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

            if (!substitutionMap.containsKey(Character.toLowerCase(ch))) {
                substitutionMap.put(Character.toLowerCase(ch), ch);
            }
        }
for (char ch = 'a'; ch <= 'z'; ch++) {
            if (!substitutionMap.containsKey(ch)) {
                substitutionMap.put(ch, ch);
            }
        }

        return substitutionMap;
    }
}
```

II. 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 – LAB2]

[OBSERVATION SPACE – LAB2]

[OBSERVATION SPACE – LAB2]

[OBSERVATION SPACE – LAB2]

[OBSERVATION SPACE – LAB2]

[OBSERVATION SPACE – LAB2]

LAB NO: 3 **Date:**

Traditional Cipher-I

Objectives:

In this lab, student will be able to:

- Understand Historical Ciphers
- Understand Encryption and Decryption

Description: The objectives of a Traditional Cipher lab typically involve introducing students to the fundamental concepts of classical or traditional encryption techniques. These objectives aim to provide students with a foundational understanding of how historical encryption methods work and their vulnerabilities.

I. SOLVED EXERCISE:

1) 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();
  }
}
```

II. 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 – LAB3]

[OBSERVATION SPACE – LAB3]

[OBSERVATION SPACE – LAB3]

[OBSERVATION SPACE – LAB3]

[OBSERVATION SPACE – LAB3]

[OBSERVATION SPACE – LAB3]

LAB NO: 4 **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');
    }
}
```

II. 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.

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

[OBSERVATION SPACE – LAB4]

LAB NO: 5 **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();
    }
}
```

II. 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 – LAB5]

[OBSERVATION SPACE – LAB5]

[OBSERVATION SPACE – LAB5]

[OBSERVATION SPACE – LAB5]

[OBSERVATION SPACE – LAB5]

[OBSERVATION SPACE – LAB5]

[OBSERVATION SPACE – LAB5]

[OBSERVATION SPACE – LAB5]

LAB NO: 6 **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 – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

[OBSERVATION SPACE – LAB6]

LAB NO: 7 **Date:**

Number Theory Algorithms

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.

Description: The Euclidean algorithm, often known as Euclid's algorithm, is an effective way to determine the greatest common divisor (GCD), or the biggest number that divides two integers (numbers) evenly and without leaving a remainder. It was initially described in the 300 BC book The Elements by the Greek mathematician Euclid, for whom it was called. One of the first algorithms still in use, an algorithm is a step-by-step process for carrying out a computation in accordance with predetermined principles. It is a component of several number-theoretic and cryptographic calculations and may be used to simplify fractions. There are several theoretical and real-world uses for the Euclidean algorithm. It is employed in conducting division in modular arithmetic as well as simplifying fractions. This algorithm is employed in both strategies for cracking these cryptosystems by factoring very big composite numbers as well as the cryptographic protocols that safeguard internet communications.

Program

```
public class EuclideanAlgorithm {
    public static int findGCD(int a, int b) {
        // Ensure that 'a' is always greater than or equal to 'b'
        if (b > a) {
            int temp = a;
            a = b;
            b = temp;
        }
while (b != 0) {
            int remainder = a % b;
            a = b;
            b = remainder;
        }

        return a;
    }
public static void main(String[] args) {
        int num1 = 36;
        int num2 = 48;
        int gcd = findGCD(num1, num2);
        System.out.println("GCD of " + num1 + " and " + num2 + " is: " + gcd);
    }
}
```

II. LAB EXERCISES:

1) Write a Java method to find the GCD of an array of integers using the Euclidean algorithm.
2) Implement the Sieve of Eratosthenes algorithm in Java to find all prime numbers up to a given limit.

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

LAB NO: 8 **Date:**

Number Theory and Applications - II

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);
    }
}
```

II. LAB EXERCISES:

1) Write a Java program to solve linear congruences using the extended Euclidean algorithm.
2) Write a Java function to compute the modular exponentiation of two numbers.

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

LAB NO: 9 **Date:**

Homomorphic Encryption Techniques

Objectives:

In this lab, student will be able to:

- Understanding Homomorphic Encryption Schemes
- Implementing Basic Homomorphic Operations
- Evaluate Performance and Efficiency

Description: Homomorphic encryption is a specialized cryptographic technique that allows computations to be performed directly on encrypted data without the need to decrypt it first. This property is known as homomorphism, and it enables secure processing of sensitive information while keeping it encrypted throughout the entire computation process. Homomorphic encryption has significant implications for privacy and security, especially in scenarios where data needs to be processed in untrusted environments, such as cloud computing and outsourced data processing.

I. SOLVED EXERCISE:

1) Write a C function that takes two integer arrays (A and B) and their respective sizes (n), and performs addition on the encrypted data using the Paillier homomorphic encryption scheme. Assume that the encryption and decryption functions are already provided.

Description: Paillier homomorphic encryption is a specific type of partially homomorphic encryption (PHE) scheme, introduced by Pascal Paillier in 1999. It is based on number theory and allows for efficient homomorphic operations on encrypted data, specifically addition of ciphertexts and multiplication of ciphertexts by a plaintext constant.

Program

```
#include <stdio.h>
#include <stdlib.h>
int paillier_encrypt(int plaintext) {
   return plaintext; // Replace this with actual encryption logic.
}
int paillier_decrypt(int ciphertext) {
   return ciphertext; // Replace this with actual decryption logic.
}
void paillier_addition(int *encrypted_result, const int *encrypted_A, const int *encrypted_B,
int n) {
   for (int i = 0; i < n; i++) {
       int decrypted_A = paillier_decrypt(encrypted_A[i]);
       int decrypted_B = paillier_decrypt(encrypted_B[i]);
       int sum = decrypted_A + decrypted_B;
       encrypted_result[i] = paillier_encrypt(sum);
   }
}
```

```
int main() {
    // Example usage
    int A[] = {3, 5, 8};
    int B[] = {7, 2, 4};
    int n = sizeof(A) / sizeof(A[0]);
  int *encrypted_A = (int *)malloc(n * sizeof(int));
    int *encrypted_B = (int *)malloc(n * sizeof(int));
    int *encrypted_result = (int *)malloc(n * sizeof(int));

    // Encrypt the data
    for (int i = 0; i < n; i++) {
        encrypted_A[i] = paillier_encrypt(A[i]);
        encrypted_B[i] = paillier_encrypt(B[i]);
    }

    // Perform addition on encrypted data
    paillier_addition(encrypted_result, encrypted_A, encrypted_B, n);

    // Print the encrypted result (just for demonstration)
    printf("Encrypted Result: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", encrypted_result[i]);
    }
    printf("\n");

    // Don't forget to free the dynamically allocated memory
    free(encrypted_A);
    free(encrypted_B);
    free(encrypted_result);
```

II. LAB EXERCISES:

1) Implement a Java class for the fully homomorphic encryption (FHE) scheme. The class should include methods for encryption, decryption, and performing addition and multiplication operations on encrypted integer arrays. You can choose any FHE scheme you are familiar with (e.g., BGV, CKKS).

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

LAB NO: 10 **Date:**

Secure Communication

Objectives:

In this lab, student will be able to:

- Understand Cryptographic Fundamentals
- Understand the SSL/TLS protocol

Description: Secure communication is the process of transmitting information or data between parties in a way that ensures confidentiality, integrity, and authenticity. The primary objective of secure communication is to protect sensitive information from unauthorized access, tampering, or interception by malicious entities. To achieve secure communication, various cryptographic techniques and protocols are employed.

I. SOLVED EXERCISE:

1) Write a C program to establish a secure SSL/TLS connection between a client and server using OpenSSL.

Description: Creating a full SSL/TLS client-server program using OpenSSL in C can be quite involved and beyond the scope of a single code snippet. However, I can provide you with a basic outline for both the client and server parts of the program.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#define CERT_FILE "server.crt"
#define KEY_FILE "server.key"

int create_socket(int port);
SSL_CTX* init_server_CTX(void);
void load_certificates(SSL_CTX* ctx, const char* cert_file, const char* key_file);
void handle_client(SSL* ssl);
int main() {
    SSL_CTX* ctx;
    int server_sock, client_sock;
    struct sockaddr_in client_addr;
    socklen_t addr_len = sizeof(client_addr);

    SSL_library_init();
    ctx = init_server_CTX();
    load_certificates(ctx, CERT_FILE, KEY_FILE);

    server_sock = create_socket(8888);
```

```
while (1) {
        client_sock = accept(server_sock, (struct sockaddr*)&client_addr, &addr_len);

        if (client_sock < 0) {
            perror("accept");
            continue;
        }
SSL* ssl = SSL_new(ctx);
        SSL_set_fd(ssl, client_sock);

        if (SSL_accept(ssl) <= 0) {
            perror("SSL_accept");
            close(client_sock);
            SSL_free(ssl);
            continue;
        }
handle_client(ssl);
        SSL_free(ssl);
        close(client_sock);
    }

    close(server_sock);
    SSL_CTX_free(ctx);

    return 0;
}
int create_socket(int port) {
    int sockfd;
    struct sockaddr_in server_addr;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port);
    if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) != 0) {
        perror("bind");
        exit(EXIT_FAILURE);
    }

    if (listen(sockfd, 5) != 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }
return sockfd;
}

SSL_CTX* init_server_CTX(void) {
    SSL_CTX* ctx;

    OpenSSL_add_all_algorithms();
    SSL_load_error_strings();
    ctx = SSL_CTX_new(SSLv23_server_method());
    if (ctx == NULL) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
    return ctx;
}

void load_certificates(SSL_CTX* ctx, const char* cert_file, const char* key_file) {
    if (SSL_CTX_use_certificate_file(ctx, cert_file, SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    if (SSL_CTX_use_PrivateKey_file(ctx, key_file, SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
}
void handle_client(SSL* ssl) {
    const char* response = "Hello, client!";

    SSL_write(ssl, response, strlen(response));
}
```

Client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#define SERVER_ADDRESS "127.0.0.1"
#define SERVER_PORT 8888
SSL_CTX* init_client_CTX(void);
void handle_server(SSL* ssl);

int main() {
    SSL_CTX* ctx;
    int client_sock;
    struct sockaddr_in server_addr;

    SSL_library_init();
    ctx = init_client_CTX();

    client_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (client_sock < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }
memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, SERVER_ADDRESS, &server_addr.sin_addr) <= 0) {
perror("inet_pton");
        exit(EXIT_FAILURE);
    }

    if (connect(client_sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) != 0) {
        perror("connect");
        exit(EXIT_FAILURE);
    }
    SSL* ssl = SSL_new(ctx);
    SSL_set_fd(ssl, client_sock);

    if (SSL_connect(ssl) <= 0) {
        perror("SSL_connect");
        close(client_sock);
        SSL_free(ssl);
        exit(EXIT_FAILURE);
    }

    handle_server(ssl);

    SSL_shutdown(ssl);
    SSL_free(ssl);
    close(client_sock);
    SSL_CTX_free(ctx);

    return 0;
}
SSL_CTX* init_client_CTX(void) {
    SSL_CTX* ctx;
    SSL_load_error_strings();
    OpenSSL_add_all_algorithms();
    ctx = SSL_CTX_new(SSLv23_client_method());

    if (ctx == NULL) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    return ctx;
}
void handle_server(SSL* ssl) {
    char buffer[1024];
    int bytes;

    bytes = SSL_read(ssl, buffer, sizeof(buffer) - 1);
    if (bytes > 0) {
        buffer[bytes] = '\0';
        printf("Received: %s\n", buffer);
    } else {
        perror("SSL_read");
    }
}
```

II. LAB EXERCISES:

1). Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm and analyse its time efficiency. Obtain the experimental results for order of growth and plot the result.

2). Write a program to implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem for any given graph and analyse its time efficiency. Obtain the experimental results for order of growth and plot the result.

3). Write a program to implement 0/1 Knapsack problem using bottom-up dynamic programming

III. ADDITIONAL EXERCISES:

1) Create a C function that takes a plaintext message and a symmetric key, then encrypts the message using AES-256.
2) Write a Java method to decrypt an incoming encrypted message using the same AES-256 symmetric key.

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]