Course Name: Cryptography & Network Security

(Code: BCSE0071)

Semester: 8th **Section: A-3 (Batch)**

Academic Session: 2023-2024

Lab File

Submitted by:- Vaibhav Varshney

Section:- A3

Roll no.:- 70

University Roll No.:- 201500766

Submitted to:- Dr. Asheesh Tiwari

INDEX

Sr. No.	Experiments	Signature
1.	Program to implement additive cipher.	
2.	Program to implement multiplicative cipher.	
3.	Program to implement Affine Cipher.	
4.	Program to implement Autokey Cipher.	
5.	Program to implement Playfair Cipher.	
6.	Program to implement RSA Algorithm.	
7.	Program to find Primitive roots.	
8.	Program to implement ElGamal encryption algorithm.	
9.	Program to implement Diffie-Hellman key generation.	
10.	Program to find Multiplicative Inverse.	
11.	Program to implement DSA Signature.	
12.	Program to implement DES Encryption.	

EXPERIMENT-1

Objective :- Implement Additive Cipher.

```
import java.util.Scanner;

public class Main {

    public static boolean isValidTextE(String text) {
        return text.matches("[a-z ]+");
    }

    public static boolean isValidTextD(String text) {
        return text.matches("[A-Z]+");
    }

    public static boolean isValidKey(String key) {
        return key.matches("\\d+") && Integer.parseInt(key) > 0;
    }

    public static String encrypt(String text, int key) {
        StringBuilder encryptedText = new StringBuilder();
        for (char ch : text.toCharArray()) {
            if (Character.isLetter(ch)) {
                char encryptedChar = (char) (((ch - 'a' + key) % 26) + 'a');
                encryptedText.append(encryptedChar);
            } else {
                encryptedText.append(ch);
            }
        }
        return encryptedText.toString().toUpperCase();
    }



     public static String decrypt(String text, int key) {
        StringBuilder decryptedText = new StringBuilder();
        for (char ch : text.toCharArray()) {
            if (Character.isLetter(ch)) {
                char decryptedChar = (char) (((ch - 'A' - key + 26) % 26) + 'A');
                decryptedText.append(decryptedChar);
            } else {
                decryptedText.append(ch);
```

```
        }
    }
    return decryptedText.toString().toLowerCase();
}

public static int bruteForceDecrypt(String Ctext, String Ptext) {
    System.out.println("Brute Force Decryption Results:");
  for (int key = 1; key < 26; key ++) {
        String decryptedText = decrypt(Ctext, key);
        if (decryptedText.equals(Ptext)) {
            return key;
        }
    }
    return -1;
}

  public static String inputTextE(Scanner scanner) {
    String plaintext;
  do {
        System.out.print("Enter plaintexttext (lowercase letters only): ");
        plaintext = scanner.nextLine();

        if (isValidTextE(plaintext)) {
            return plaintext;
        } else {
            System.out.println("Invalid input.");
        }
    } while (true);

}

public static String inputTextD(Scanner scanner) {
    String ciphertext;
    do {
        System.out.print("Enter ciphertext (uppercase letters only): ");
        ciphertext = scanner.nextLine();

        if (isValidTextD(ciphertext)) {
            return ciphertext;
        } else {
            System.out.println("Invalid input.");
        }
    } while (true);
}

public static int inputKey(Scanner scanner) {
    int key;
    do {
        System.out.print("Enter the key (0-25): ");
        String keyInput = scanner.nextLine();
        if (isValidKey(keyInput)) {
```

```
            key = Integer.parseInt(keyInput);
            break;
        } else {
            System.out.println("Invalid key input. Please enter a number between 0 and 25.");
        }
    } while (true);
    return key;
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Encryption");
        System.out.println("2. Decryption");
        System.out.println("3. Brute Force Decryption");
        System.out.println("4. Exit");

        System.out.print("Enter your choice (1-4): ");
        String choice = scanner.nextLine();

        switch (choice) {
            case "1":
                String plaintext = inputTextE(scanner);
                int key = inputKey(scanner);
                String encryptedText = encrypt(plaintext, key);
                System.out.println("Encrypted text: " + encryptedText);
                break;

            case "2":
                 String ciphertext = inputTextD(scanner);
                 key = inputKey(scanner);
                 System.out.println(ciphertext + " " + key);
                String decryptedText = decrypt(ciphertext, key);
                System.out.println("Decrypted text: " + decryptedText);
                break;

            case "3":
                 String bruteForceCiphertext = inputTextD(scanner);
                  String bruteForcePlaintext = inputTextE(scanner);
                int ans  = bruteForceDecrypt(bruteForceCiphertext,bruteForcePlaintext);
                System.out.println("Matched key " + ans);
                break;

            case "4":
                System.out.println("Exiting the program.");
                System.exit(0);

            default:
                System.out.println("Invalid choice. Please enter a number between 1 and 4.");
        }
    }
```

```
    }
}
```

EXPERIMENT-2

Objective :- Implement Multiplicative Cipher.

```
import java.util.Scanner;

public class Main {

    public static boolean isValidTextE(String text) {
        return text.matches("[a-z ]+");
    }

    public static boolean isValidTextD(String text) {
        return text.matches("[A-Z ]+");
    }

    public static boolean isValidKey(String key) {
        return key.matches("\\d+") && Integer.parseInt(key) % 2 != 0 && Integer.parseInt(key)!=13
&& Integer.parseInt(key) > 1 && Integer.parseInt(key) < 26;
    }

     public static int modInverse(int a, int m) {
        for (int i = 1; i < m; i++) {
            if ((a * i) % m == 1) {
                return i;
            }
        }
        return -1;
    }

    public static String encrypt(String text, int key) {

        StringBuilder encryptedText = new StringBuilder();
        for (char ch : text.toCharArray()) {
            if (Character.isLetter(ch)) {
                char encryptedChar = (char) ((ch - 'a') * key % 26 + 'a');
                encryptedText.append(encryptedChar);
            } else {
                encryptedText.append(ch);
            }
        }
        return encryptedText.toString().toUpperCase();
    }
```

```
 public static String decrypt(String text, int key) {

    int inverseKey = modInverse(key, 26);

    StringBuilder decryptedText = new StringBuilder();

    for (char ch : text.toCharArray()) {
        if (Character.isLetter(ch)) {
            decryptedText.append((char) ((ch - 'A') * inverseKey % 26 + 'A'));
        } else {
            decryptedText.append(ch);
        }
    }

    return decryptedText.toString().toLowerCase();
}

public static int bruteForceDecrypt(String Ctext, String Ptext) {
    System.out.println("Brute Force Decryption Results:");
  for (int key = 1; key < 26; key ++) {
        String decryptedText = decrypt(Ctext, key);
        if (decryptedText.equals(Ptext)) {
            return key;
        }
    }
    return -1;
}


  public static String inputTextE(Scanner scanner) {
    String plaintext;
  do {
        System.out.print("Enter plaintexttext (lowercase letters only): ");
        plaintext = scanner.nextLine();

        if (isValidTextE(plaintext)) {
            return plaintext;
        } else {
            System.out.println("Invalid input.");
        }
    } while (true);
}

public static String inputTextD(Scanner scanner) {
    String ciphertext;
    do {
        System.out.print("Enter ciphertext (uppercase letters only): ");
        ciphertext = scanner.nextLine();
```

```
        if (isValidTextD(ciphertext)) {
            return ciphertext;
        } else {
            System.out.println("Invalid input.");
        }
    } while (true);
}

public static int inputKey(Scanner scanner) {
    int key;
    do {
        System.out.print("Enter the key (1-25): ");
        String keyInput = scanner.nextLine();
        if (isValidKey(keyInput)) {
            key = Integer.parseInt(keyInput);
            break;
        } else {
            System.out.println("Invalid key input. Please enter a number between 1 and 25.");
        }
    } while (true);
    return key;
}

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

    while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Encryption");
        System.out.println("2. Decryption");
        System.out.println("3. Brute Force Decryption");
        System.out.println("4. Exit");

        System.out.print("Enter your choice (1-4): ");
        String choice = scanner.nextLine();

        switch (choice) {
            case "1":
                String plaintext = inputTextE(scanner);
                int key = inputKey(scanner);
                String encryptedText = encrypt(plaintext, key);
                System.out.println("Encrypted text: " + encryptedText);
                break;

            case "2":
                 String ciphertext = inputTextD(scanner);
                 key = inputKey(scanner);
                 System.out.println(ciphertext + " " + key);
                String decryptedText = decrypt(ciphertext, key);
                System.out.println("Decrypted text: " + decryptedText);
```

```
                break;

            case "3":
                String bruteForceCiphertext = inputTextD(scanner);
                 String bruteForcePlaintext = inputTextE(scanner);
                int ans  = bruteForceDecrypt(bruteForceCiphertext,bruteForcePlaintext);
                System.out.println("Matched key " + ans);
                break;


            case "4":
                System.out.println("Exiting the program.");
                System.exit(0);

            default:
                System.out.println("Invalid choice. Please enter a number between 1 and 4.");
        }
    }
}
```

EXPERIMENT-3

Objective :- Implement Affine Cipher.

```
import java.util.Scanner;

public class Main {

    public static boolean isValidTextE(String text) {
        return text.matches("[a-z ]+");
    }

    public static boolean isValidTextD(String text) {
        return text.matches("[A-Z ]+");
    }

    public static boolean isValidKey1(String key) {


        return key.matches("\\d+") &&
           Integer.parseInt(key) % 2 != 0 && Integer.parseInt(key)  != 13 &&
           Integer.parseInt(key)  > 1 && Integer.parseInt(key)  < 26 &&
           isCoprime(Integer.parseInt(key) , 26);
    }

    public static boolean isValidKey2(String key) {
        //return key.matches("\\d+") && Integer.parseInt(key) > 0 && Integer.parseInt(key) < 26;
        return key.matches("\\d+") && Integer.parseInt(key) > 0 && Integer.parseInt(key) < 26;
    }



    public static boolean isCoprime(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a == 1;
    }

    public static int modInverse(int a, int m) {
        for (int i = 1; i < m; i++) {
            if ((a * i) % m == 1) {
                return i;
```

```
            }
        }
        return -1;
    }

    public static String encrypt(String text, int a, int b) {
    StringBuilder encryptedText = new StringBuilder();
    for (char ch : text.toCharArray()) {
        if (Character.isLetter(ch)) {
            char encryptedChar = (char) (((ch - 'a') * a + b) % 26 + 'a');
            encryptedText.append(encryptedChar);
        } else {
            encryptedText.append(ch);
        }
    }
    return encryptedText.toString().toUpperCase();
}

    public static String decrypt(String text, int a, int b) {
        int inverseKey = modInverse(a, 26);

        if (inverseKey == -1 || !isCoprime(a, 26)) {
            System.out.println("Invalid key. Modular inverse does not exist.");
            return "";
        }

        StringBuilder decryptedText = new StringBuilder();

        for (char ch : text.toCharArray()) {
            if (Character.isLetter(ch)) {
                char decryptedChar = (char) (((inverseKey * (ch - 'A' - b) % 26) + 26) % 26 + 'A');
                decryptedText.append(decryptedChar);
            } else {
                decryptedText.append(ch);
            }
        }

        return decryptedText.toString().toLowerCase();
    }

    public static int bruteForceDecrypt(String Ctext, String Ptext) {
        System.out.println("Brute Force Decryption Results:");
        for (int a = 1; a < 26; a++) {
            if (isCoprime(a, 26)) {
                for (int b = 0; b < 26; b++) {
                    String decryptedText = decrypt(Ctext, a, b);
                    if (decryptedText.equals(Ptext)) {
                        return a * 100 + b; // Encoding both 'a' and 'b' in a single integer for simplicity
                    }
                }
            }
```

```
    }
    return -1;
}

public static String inputTextE(Scanner scanner) {
    String plaintext;
    do {
        System.out.print("Enter plaintext (lowercase letters only): ");
        plaintext = scanner.nextLine();

        if (isValidTextE(plaintext)) {
            return plaintext;
        } else {
            System.out.println("Invalid input.");
        }
    } while (true);
}

public static String inputTextD(Scanner scanner) {
    String ciphertext;
    do {
        System.out.print("Enter ciphertext (uppercase letters only): ");
        ciphertext = scanner.nextLine();

        if (isValidTextD(ciphertext)) {
            return ciphertext;
        } else {
            System.out.println("Invalid input.");
        }
    } while (true);
}

public static int inputKey1(Scanner scanner) {
    int key;
    do {
        System.out.print("Enter the key coprime with 26: ");
        String keyInput = scanner.nextLine();
        if (isValidKey1(keyInput)) {
            key = Integer.parseInt(keyInput);
            break;
        } else {
            System.out.println("Invalid key input. Please enter a coprime number.");
        }
    } while (true);
    return key;
}

 public static int inputKey2(Scanner scanner) {
    int key;
    do {
        System.out.print("Enter the key (1-25): ");
```

```
        String keyInput = scanner.nextLine();
        if (isValidKey2(keyInput)) {
            key = Integer.parseInt(keyInput);
            break;
        } else {
            System.out.println("Invalid key input. Please enter a number between 1 and 25.");
        }
    } while (true);
    return key;
}

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

    while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Encryption");
        System.out.println("2. Decryption");
        System.out.println("3. Brute Force Decryption");
        System.out.println("4. Exit");

        System.out.print("Enter your choice (1-4): ");
        String choice = scanner.nextLine();

        switch (choice) {
            case "1":
                String plaintext = inputTextE(scanner);
                int a = inputKey1(scanner);
                int b = inputKey2(scanner);
                String encryptedText = encrypt(plaintext, a, b);
                System.out.println("Encrypted text: " + encryptedText);
                break;

            case "2":
                String ciphertext = inputTextD(scanner);
                a = inputKey1(scanner);
                b = inputKey2(scanner);
                String decryptedText = decrypt(ciphertext, a, b);
                System.out.println("Decrypted text: " + decryptedText);
                break;

            case "3":
                String bruteForceCiphertext = inputTextD(scanner);
                String bruteForcePlaintext = inputTextE(scanner);
                int ans = bruteForceDecrypt(bruteForceCiphertext, bruteForcePlaintext);
                if (ans != -1) {
                    int foundA = ans / 100;
                    int foundB = ans % 100;
                    System.out.println("Found key: a=" + foundA + ", b=" + foundB);
                } else {
                    System.out.println("Key not found.");
```

```
                }
                break;

            case "4":
                System.out.println("Exiting the program.");
                System.exit(0);

            default:
                System.out.println("Invalid choice. Please enter a number between 1 and 4.");
        }
    }
}
}
```

EXPERIMENT-4

Objective :- Implement Auto Key Cipher.

```
import java.util.*;

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

      int choice;

      do {
         try {
            System.out.println("Enter your choice:");
            System.out.println("1. Encrypt using Autokey Cipher");
            System.out.println("2. Decrypt using Autokey Cipher");
            System.out.println("3. Brute Force Decrypt");
            System.out.println("0. Exit");

            choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
               case 1:
                  performEncryption(scanner);
                  break;
               case 2:
                  performDecryption(scanner);
                  break;
               case 3:
                  performBruteForce(scanner);
                  break;
               case 0:
                  System.out.println("Exiting the program");
                  break;
               default:
                  System.out.println("Invalid choice. Please enter a valid option.");
            }
         } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a valid option.");
            scanner.nextLine(); // Clear the input buffer
            choice = -1; // Reset choice to force re-prompting
         }
      } while (choice != 0);
```

```
        scanner.close();
    }

    public static void performEncryption(Scanner scanner) {
        System.out.println("Enter the plaintext:");
        String plaintext = scanner.nextLine().toLowerCase().replaceAll("[^a-z]", ""); // Convert to
lowercase and remove non-letter characters

        System.out.println("Enter the numeric key:");
        int key = scanner.nextInt();

        String encryptedText = encryptAutokeyCipher(plaintext, key);

        System.out.println("Encrypted Text: " + encryptedText);
    }

    public static void performDecryption(Scanner scanner) {
        System.out.println("Enter the ciphertext:");
        String ciphertext = scanner.nextLine().toLowerCase().replaceAll("[^a-z]", ""); // Convert to
lowercase and remove non-letter characters

        System.out.println("Enter the numeric key:");
        int key = scanner.nextInt();

        String decryptedText = decryptAutokeyCipher(ciphertext, key);

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

    public static String encryptAutokeyCipher(String plaintext, int key) {
        StringBuilder encryptedText = new StringBuilder();

        for (int i = 0; i < plaintext.length(); i++) {
            char plainChar = plaintext.charAt(i);
            if (Character.isLetter(plainChar)) {
                char keyChar = (char) ((plainChar - 'a' + key) % 26 + 'a');
                encryptedText.append(keyChar);
                key = plainChar - 'a'; // Update (t-a= 19)
            } else {
                encryptedText.append(plainChar);
            }
        }

        return encryptedText.toString();
    }

    public static String decryptAutokeyCipher(String ciphertext, int key) {
        StringBuilder decryptedText = new StringBuilder();

        for (int i = 0; i < ciphertext.length(); i++) {
            char cipherChar = ciphertext.charAt(i);
```

```
            if (Character.isLetter(cipherChar)) {
                char keyChar = (char) ((cipherChar - 'a' - key + 26) % 26 + 'a');
                decryptedText.append(keyChar);
                key = keyChar - 'a'; //update
            } else {
                decryptedText.append(cipherChar);
            }
        }

        return decryptedText.toString();
    }

    public static void performBruteForce(Scanner scanner) {
        System.out.println("Enter the ciphertext:");
        String ciphertext = scanner.nextLine().toLowerCase().replaceAll("[^a-z]", "");

        System.out.println("Enter the corresponding plaintext:");
        String plaintext = scanner.nextLine().toLowerCase().replaceAll("[^a-z]", "");

        for (int key = 0; key < 26; key++) {
            String decryptedText = decryptAutokeyCipher(ciphertext, key);
            if (decryptedText.equals(plaintext)) {
                System.out.println("Key found: " + key);
                return;
            }
        }

        System.out.println("Key not found.");
    }
}
```

EXPERIMENT-5

Objective :- Implement Playfair Cipher.

```
import java.util.Scanner;
public class PlayfairCipher {
   private static final int MATRIX_SIZE = 5;
   private static final char[][] PLAYFAIR_MATRIX = new
char[MATRIX_SIZE][MATRIX_SIZE];
   private static String key;
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter the key (letters only, no spaces or special characters):");
      key = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");
      generatePlayfairMatrix();
      while (true) {
         System.out.println("Enter your choice:");
         System.out.println("1. Encrypt");
         System.out.println("2. Decrypt");
         System.out.println("3. Exit");
         int choice = scanner.nextInt();
         scanner.nextLine(); // Consume newline
         switch (choice) {
            case 1:
               System.out.println("Enter the plaintext:");
               String plaintext = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");
               System.out.println("Encrypted Text: " + encrypt(plaintext));
               break;
            case 2:
               System.out.println("Enter the ciphertext:");
               String ciphertext = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");
               System.out.println("Decrypted Text: " + decrypt(ciphertext));
               break;
            case 3:
               System.out.println("Exiting...");
               return;
            default:
               System.out.println("Invalid choice. Please enter 1, 2, or 3.");
               break;
         }
      }
   }
   private static void generatePlayfairMatrix() {
      String keyWithoutDuplicates = removeDuplicateCharacters(key +
"ABCDEFGHIJKLMNOPQRSTUVWXYZ");
```

```
        int index = 0;
        for (int i = 0; i < MATRIX_SIZE; i++) {
            for (int j = 0; j < MATRIX_SIZE; j++) {
                PLAYFAIR_MATRIX[i][j] = keyWithoutDuplicates.charAt(index);
                index++;
            }
        }
    }
    private static String removeDuplicateCharacters(String str) {
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < str.length(); i++) {
            if (result.indexOf(String.valueOf(str.charAt(i))) == -1) {
                result.append(str.charAt(i));
            }
        }
        return result.toString();
    }

    private static String encrypt(String plaintext) {
        StringBuilder encryptedText = new StringBuilder();

        // Prepare the plaintext (remove non-alphabetic characters, convert to uppercase)
        plaintext = plaintext.replaceAll("[^A-Z]", "").toUpperCase();

        // Add a dummy character if the length of plaintext is odd
        if (plaintext.length() % 2 != 0) {
            plaintext += 'X';
        }

        // Encrypt each digram
        for (int i = 0; i < plaintext.length(); i += 2) {
            char firstChar = plaintext.charAt(i);
            char secondChar = plaintext.charAt(i + 1);

            int[] firstPosition = findPosition(firstChar);
            int[] secondPosition = findPosition(secondChar);

            // Apply Playfair Cipher rules
            if (firstPosition[0] == secondPosition[0]) { // Same row
                encryptedText.append(PLAYFAIR_MATRIX[firstPosition[0]][(firstPosition[1] + 1) %
MATRIX_SIZE]);
                encryptedText.append(PLAYFAIR_MATRIX[secondPosition[0]][(secondPosition[1] + 1)
% MATRIX_SIZE]);
            } else if (firstPosition[1] == secondPosition[1]) { // Same column
                encryptedText.append(PLAYFAIR_MATRIX[(firstPosition[0] + 1) %
MATRIX_SIZE][firstPosition[1]]);
                encryptedText.append(PLAYFAIR_MATRIX[(secondPosition[0] + 1) %
MATRIX_SIZE][secondPosition[1]]);
            } else { // Rectangle
                encryptedText.append(PLAYFAIR_MATRIX[firstPosition[0]][secondPosition[1]]);
                encryptedText.append(PLAYFAIR_MATRIX[secondPosition[0]][firstPosition[1]]);
```

```
            }
        }

        return encryptedText.toString();
    }

    private static String decrypt(String ciphertext) {
        StringBuilder decryptedText = new StringBuilder();

        // Prepare the ciphertext (remove non-alphabetic characters, convert to uppercase)
        ciphertext = ciphertext.replaceAll("[^A-Z]", "").toUpperCase();

        // Decrypt each digram
        for (int i = 0; i < ciphertext.length(); i += 2) {
            char firstChar = ciphertext.charAt(i);
            char secondChar = ciphertext.charAt(i + 1);

            int[] firstPosition = findPosition(firstChar);
            int[] secondPosition = findPosition(secondChar);

            // Apply Playfair Cipher rules (reverse)
            if (firstPosition[0] == secondPosition[0]) { // Same row
                decryptedText.append(PLAYFAIR_MATRIX[firstPosition[0]][(firstPosition[1] +
MATRIX_SIZE - 1) % MATRIX_SIZE]);
                decryptedText.append(PLAYFAIR_MATRIX[secondPosition[0]][(secondPosition[1] +
MATRIX_SIZE - 1) % MATRIX_SIZE]);
            } else if (firstPosition[1] == secondPosition[1]) { // Same column
                decryptedText.append(PLAYFAIR_MATRIX[(firstPosition[0] + MATRIX_SIZE - 1) %
MATRIX_SIZE][firstPosition[1]]);
                decryptedText.append(PLAYFAIR_MATRIX[(secondPosition[0] + MATRIX_SIZE - 1)
% MATRIX_SIZE][secondPosition[1]]);
            } else { // Rectangle
                decryptedText.append(PLAYFAIR_MATRIX[firstPosition[0]][secondPosition[1]]);
                decryptedText.append(PLAYFAIR_MATRIX[secondPosition[0]][firstPosition[1]]);
            }
        }
        return decryptedText.toString();
    }
    // Helper method to find the position of a character in the Playfair matrix
    private static int[] findPosition(char ch) {
        int[] position = new int[2];
        for (int i = 0; i < MATRIX_SIZE; i++) {
            for (int j = 0; j < MATRIX_SIZE; j++) {
                if (PLAYFAIR_MATRIX[i][j] == ch) {
                    position[0] = i;
                    position[1] = j;
                    return position;
                }
            }
        }
        return position;
```

```
    }
}
```

EXPERIMENT-6

Objective :- Implement RSA algorithm.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class RSA {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("RSA Encryption/Decryption");
        System.out.println("==========================");
        System.out.println("1. Encrypt a message");
        System.out.println("2. Decrypt a message");
        System.out.println("3. Exit");

        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        if (choice == 1) {
            System.out.println("Encryption");
            System.out.println("==========");
            encryptionMenu(scanner);
        } else if (choice == 2) {
            System.out.println("Decryption");
            System.out.println("==========");
            decryptionMenu(scanner);
        } else if (choice == 3) {
            System.out.println("Exiting...");
            return;
        } else {
            System.out.println("Invalid choice. Exiting...");
            return;
        }
        scanner.close();
    }

    private static void encryptionMenu(Scanner scanner) {
        System.out.println("Enter the value of p (prime number):");
        int p = scanner.nextInt();
        System.out.println("Enter the value of q (prime number):");
        int q = scanner.nextInt();

        // Check if p and q are prime numbers
        if (!isPrime(p) || !isPrime(q)) {
            System.out.println("Both p and q must be prime numbers.");
```

```
            return;
        }

        if(p==q)
        {
            System.out.println("p and q are equal");
            return;
        }

        // Calculate n and phi(n)
        int n = p * q;
        int phi = (p - 1) * (q - 1);

        List<int[]> keyPairs = generateKeyPairs(phi);

        System.out.println("Available (e, d) pairs:");
        for (int i = 0; i < keyPairs.size(); i++) {
            int[] pair = keyPairs.get(i);
            System.out.println((i + 1) + ". (e, d) = (" + pair[0] + ", " + pair[1] + ")");
        }

        System.out.println("Enter the index of the chosen pair:");
        int pairIndex = scanner.nextInt();
        if (pairIndex < 1 || pairIndex > keyPairs.size()) {
            System.out.println("Invalid pair index.");
            return;
        }

        int[] chosenPair = keyPairs.get(pairIndex - 1);
        int e = chosenPair[0];
        int d = chosenPair[1];

        System.out.println("Public Key (n, e): (" + n + ", " + e + ")");
        System.out.println("Private Key (n, d): (" + n + ", " + d + ")");

        scanner.nextLine(); // Consume newline

        System.out.println("Enter the plaintext (integer value):");
        int plaintext = scanner.nextInt();

        if(plaintext>n){
            System.out.println("plaintext should be less than n");
            return;
        }

        // Encrypt plaintext
        int ciphertext = encrypt(plaintext, n, e);
        System.out.println("Ciphertext: " + ciphertext);
    }

    private static void decryptionMenu(Scanner scanner) {
```

```
        System.out.println("Enter the value of p (prime number):");
        int p = scanner.nextInt();
        System.out.println("Enter the value of q (prime number):");
        int q = scanner.nextInt();

        // Check if p and q are prime numbers
        if (!isPrime(p) || !isPrime(q)) {
            System.out.println("Both p and q must be prime numbers.");
            return;
        }

        // Calculate n and phi(n)
        int n = p * q;
        int phi = (p - 1) * (q - 1);

        List<int[]> keyPairs = generateKeyPairs(phi);

        System.out.println("Available (e, d) pairs:");
        for (int i = 0; i < keyPairs.size(); i++) {
            int[] pair = keyPairs.get(i);
            System.out.println((i + 1) + ". (e, d) = (" + pair[0] + ", " + pair[1] + ")");
        }

        System.out.println("Enter the index of the chosen pair:");
        int pairIndex = scanner.nextInt();
        if (pairIndex < 1 || pairIndex > keyPairs.size()) {
            System.out.println("Invalid pair index.");
            return;
        }

        int[] chosenPair = keyPairs.get(pairIndex - 1);
        int e = chosenPair[0];
        int d = chosenPair[1];

        System.out.println("Public Key (n, e): (" + n + ", " + e + ")");
        System.out.println("Private Key (n, d): (" + n + ", " + d + ")");

        scanner.nextLine(); // Consume newline

        System.out.println("Enter the ciphertext (integer value):");
        int ciphertext = scanner.nextInt();

        // Decrypt ciphertext
        int plaintext = decrypt(ciphertext, n, d);
        System.out.println("Decrypted Text (plaintext): " + plaintext);
    }

    // Check if a number is prime
    private static boolean isPrime(int num) {
        if (num <= 1) return false;
        if (num <= 3) return true;
```

```
    if (num % 2 == 0 || num % 3 == 0) return false;
    for (int i = 5; i * i <= num; i += 6) {
        if (num % i == 0 || num % (i + 2) == 0) return false;
    }
    return true;
}

// Generate (e, d) pairs
private static List<int[]> generateKeyPairs(int phi) {
    List<int[]> keyPairs = new ArrayList<>();
    for (int e = 2; e < phi; e++) {
        if (gcd(e, phi) == 1) {
            int d = modInverse(e, phi);
            keyPairs.add(new int[]{e, d});
        }
    }
    return keyPairs;
}

// Calculate the greatest common divisor (gcd) of two numbers
private static int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

// Calculate the modular multiplicative inverse of a number modulo m
private static int modInverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1) return x;
    }
    return -1;
}
// Encryption function
private static int encrypt(int plaintext, int n, int e) {
    return power(plaintext, e, n);
}
// Decryption function
private static int decrypt(int ciphertext, int n, int d) {
    return power(ciphertext, d, n);
}
// Modular exponentiation
private static int power(int x, int y, int p) {
    int result = 1;
    x = x % p;
    while (y > 0) {
        if (y % 2 == 1) result = (result * x) % p;
        y = y / 2;
        x = (x * x) % p;
    }
    return result;
```

```
    }
}
```

EXPERIMENT-7

Objective :- Implement program to find primitive roots.

```
import java.util.*;

public class Primitive {

    // Function to calculate gcd of two numbers
    static int gcd(int a, int b) {
        if (b == 0)
            return a;
        return gcd(b, a % b);
    }

    // Function to calculate φ(n) - Euler's totient function
    static int totient(int n) {
        int result = 1;
        for (int i = 2; i < n; i++)
            if (gcd(i, n) == 1)
                result++;
        return result;
    }

    // Function to calculate powers modulo n
    static int power(int x, int y, int p) {
        int res = 1;
        x = x % p;
        while (y > 0) {
            if (y % 2 == 1)
                res = (res * x) % p;
            y = y / 2;
            x = (x * x) % p;
        }
        return res;
    }

    // Function to check if a is primitive root of n
    static boolean isPrimitiveRoot(int a, int n) {
        if (n == 2)
            return true; // Special case for n = 2
        int phi = totient(n);
        for (int i = 2; i < phi; i++)
            if (power(a, i, n) == 1)
                return false;
        return true;
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input;
        while (true) {
            System.out.print("Enter a number to find its primitive roots (or 'e' to exit): ");
            input = scanner.nextLine().trim();
            if (input.equalsIgnoreCase("e")) {
                break;
            }
            try {
                int number = Integer.parseInt(input);
                int phi = totient(number);
                System.out.println("φ(" + number + ") = " + phi);
                System.out.println("Primitive roots of " + number + ": ");
                if (number == 2) {
                    System.out.println(1); // Special case for n = 2
                } else {
                    boolean foundPrimitiveRoot = false;
                    for (int a = 2; a < number; a++) {
                        if (gcd(a, number) == 1 && isPrimitiveRoot(a, number)) {
                            System.out.print(a + " ");
                            foundPrimitiveRoot = true;
                        }
                    }
                    if (!foundPrimitiveRoot)
                        System.out.println("none");
                }
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid number or 'e' to exit.");
            }
        }
        scanner.close();
    }
}
```

EXPERIMENT-8

Objective :- Implement ElGamal encryption algorithm.

```
import java.util.Scanner;
import java.util.ArrayList;

public class Elgamal {

    // Function to find primitive roots
    static int gcd(int a, int b) {
        if (b == 0)
            return a;
        return gcd(b, a % b);
    }

    // Function to calculate φ(n) - Euler's totient function
    static int totient(int n) {
        int result = 1;
        for (int i = 2; i < n; i++)
            if (gcd(i, n) == 1)
                result++;
        return result;
    }

    // Function to calculate powers modulo n
    static int power(int x, int y, int p) {
        int res = 1;
        x = x % p;
        while (y > 0) {
            if (y % 2 == 1)
                res = (res * x) % p;
            y = y / 2;
            x = (x * x) % p;
        }
        return res;
    }

    // Function to check if a is primitive root of n
    static boolean isPrimitiveRoot(int a, int n) {
        if (n == 2)
            return true; // Special case for n = 2
        int phi = totient(n);
        for (int i = 2; i < phi; i++)
            if (power(a, i, n) == 1)
                return false;
        return true;
```

```
    }

    static ArrayList<Integer> findPrimitiveRoots(int p) {
        ArrayList<Integer> primitiveRoots = new ArrayList<>();

        int number = p;
        int phi = totient(number);
        System.out.println("φ(" + number + ") = " + phi);
        if (number == 2) {
            primitiveRoots.add(1);
        } else {
            boolean foundPrimitiveRoot = false;
            for (int a = 2; a < number; a++) {
                if (gcd(a, number) == 1 && isPrimitiveRoot(a, number)) {
                    primitiveRoots.add(a);
                    foundPrimitiveRoot = true;
                }
            }
            if (!foundPrimitiveRoot)
                System.out.println("none");
        }

        return primitiveRoots;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.print("Enter a prime number (p): ");
            int p = scanner.nextInt();

            // Finding primitive roots of p
            ArrayList<Integer> primitiveRoots = findPrimitiveRoots(p);
            if (primitiveRoots.isEmpty()) {
                System.out.println("No primitive roots found for " + p + ". Please enter another prime
number.");
                continue;
            }

            System.out.println("Primitive Roots of " + p + ": " + primitiveRoots);

            System.out.print("Choose a primitive root: ");
            int primitiveRoot = scanner.nextInt();
            if (!primitiveRoots.contains(primitiveRoot)) {
                System.out.println("Invalid primitive root.");
                continue;
            }

            // Key generation

            System.out.print("Enter private key (a): ");
```

```
    int a = scanner.nextInt();
    if (a >= p || a <= 1) {
        System.out.println("Invalid private key. It should be greater than 1 and less than p.");
        continue;
    }
    int A = power(primitiveRoot, a, p);

    System.out.println("Public key (A): " + A);

    // Encryption
    int M, k;
    while (true) {
        System.out.print("Enter the message to encrypt (M): ");
        M = scanner.nextInt();
        if (M >= p) {
            System.out.println("Invalid message. It should be less than p.");
        } else {
            break;
        }
    }

    while (true) {
        System.out.print("Enter random number (k): ");
        k = scanner.nextInt();
        if (k >= p || k <= 1) {
            System.out.println("Invalid value of k. It should be greater than 1 and less than p.");
        } else {
            break;
        }
    }
    int K = power(primitiveRoot, k, p);

    int C1 = power(primitiveRoot, k, p);
    int C2 = (M * power(A, k, p)) % p;

    System.out.println("Encrypted Message (C1, C2): (" + C1 + ", " + C2 + ")")
    // Decryption
    System.out.print("Enter C1: ");
    int C1_decrypt = scanner.nextInt();
    System.out.print("Enter C2: ");
    int C2_decrypt = scanner.nextInt();
    int decryptedMessage = (C2_decrypt * power(C1_decrypt, p - 1 - a, p)) % p;
    System.out.println("Decrypted Message: " + decryptedMessage);

    // Ask user if they want to continue
    System.out.print("Do you want to continue? (y/n): ");
    String choice = scanner.next();
    if (!choice.equalsIgnoreCase("y"))
        break;
}
scanner.close();
```

```
    }
}
```

EXPERIMENT-9

Objective :- Implement Diffie-Hellman key generation.

```
import java.util.ArrayList;
import java.util.Scanner;

public class Diffie {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a prime number (p): ");
        int p = getValidPrime(scanner)
        ArrayList<Integer> primitiveRoots = findPrimitiveRoots(p);
        System.out.println("Primitive Roots of " + p + ": " + primitiveRoots);
        System.out.print("Select a primitive root (g): ");
        int g = getValidPrimitiveRoot(scanner, primitiveRoots, p);
        System.out.print("Enter Alice's private key (a): ");
        int privateKeyAlice = getValidPrivateKey(scanner, p);
        System.out.print("Enter Bob's private key (b): ");
        int privateKeyBob = getValidPrivateKey(scanner, p);
        System.out.println("Alice Private Key: " + privateKeyAlice);
        System.out.println("Bob Private Key: " + privateKeyBob);
        int publicKeyAlice = calculatePublicKey(p, g, privateKeyAlice);
        int publicKeyBob = calculatePublicKey(p, g, privateKeyBob);
        System.out.println("Alice Public Key: " + publicKeyAlice);
        System.out.println("Bob Public Key: " + publicKeyBob);
        int sharedSecretAlice = calculateSharedSecret(p, publicKeyBob, privateKeyAlice);
        int sharedSecretBob = calculateSharedSecret(p, publicKeyAlice, privateKeyBob);
        System.out.println("ALice Shared Secret Key: " + sharedSecretAlice);
        System.out.println("Bob Shared Secret Key: " + sharedSecretBob);
        scanner.close();
    }

    public static int getValidPrime(Scanner scanner) {
        int p;
        while (true) {
            if (!scanner.hasNextInt()) {
                System.out.println("Invalid input. Please enter a valid integer.");
                scanner.next();
                continue;
            }
            p = scanner.nextInt();
            if (isPrime(p)) {
                break;
            } else {
                System.out.println(p + " is not a prime number. Please enter a prime number.");
            }
```

```
        }
        return p;
    }

    public static int getValidPrimitiveRoot(Scanner scanner, ArrayList<Integer> primitiveRoots, int
p) {
        int g;
        while (true) {
            if (!scanner.hasNextInt()) {
                System.out.println("Invalid input. Please enter a valid integer.");
                scanner.next();
                continue;
            }
            g = scanner.nextInt();
            if (primitiveRoots.contains(g) && g < p) {
                break;
            } else {
                System.out.println(g + " is not a valid primitive root for " + p +
                        ". Please select a primitive root less than " + p + ".");
            }
        }
        return g;
    }

    public static int getValidPrivateKey(Scanner scanner, int p) {
        int privateKey;
        while (true) {
            if (!scanner.hasNextInt()) {
                System.out.println("Invalid input. Please enter a valid integer.");
                scanner.next();
                continue;
            }
            privateKey = scanner.nextInt();
            if (privateKey > 1 && privateKey < p) {
                break;
            } else {
                System.out.println("Private key should be greater than 1 and less than " + p);
            }
        }
        return privateKey;
    }

    public static ArrayList<Integer> findPrimitiveRoots(int p) {
        ArrayList<Integer> primitiveRoots = new ArrayList<>();
        for (int i = 1; i < p; i++) {
            if (isPrimitiveRoot(i, p)) {
                primitiveRoots.add(i);
            }
        }
        return primitiveRoots;
    }
```

```
public static boolean isPrime(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i <= Math.sqrt(n); i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

public static boolean isPrimitiveRoot(int g, int p) {
    ArrayList<Integer> factors = factorize(p - 1);
    for (int factor : factors) {
        if (power(g, (p - 1) / factor, p) == 1) {
            return false;
        }
    }
    return true;
}

public static int calculatePublicKey(int p, int g, int privateKey) {
    return power(g, privateKey, p);
}

public static int calculateSharedSecret(int p, int publicKey, int privateKey) {
    return power(publicKey, privateKey, p);
}

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

public static ArrayList<Integer> factorize(int n) {
    ArrayList<Integer> factors = new ArrayList<>();
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            factors.add(i);
            n /= i;
        }
    }
    if (n > 1) {
        factors.add(n);
    }
    return factors;
```

```
    }
}
```

EXPERIMENT-10

Objective :- Implement a program to find Multiplicative Inverse.

```
import java.util.*;
public class MultiInv {
    public static int gcdExtended(int a, int b, int[] x, int[] y) {
        if (a == 0) {
            x[0] = 0;
            y[0] = 1;
            return b;
        }
        int[] x1 = new int[1];
        int[] y1 = new int[1];
        int gcd = gcdExtended(b % a, a, x1, y1);
        x[0] = y1[0] - (b / a) * x1[0];
        y[0] = x1[0];
        return gcd;
    }

    public static void modInverse(int A, int M) {
        int[] x = new int[1];
        int[] y = new int[1];
        int g = gcdExtended(A, M, x, y);
        if (g != 1) {
            System.out.println("Inverse doesn't exist");
        } else {
            int res = (x[0] % M + M) % M;
            System.out.println("Modular multiplicative inverse is: " + res);
        }
    }

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the number whose inverse is to be found (a): ");
        int A=sc.nextInt();
         System.out.println("Enter the modulo value (m): ");
        int M=sc.nextInt();
        modInverse(A, M);
    }
}
```

EXPERIMENT-11

Objective :- Implement DSA Signature.

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

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

      // Input prime divisor q
      System.out.print("Enter the prime divisor q: ");
      BigInteger q = scanner.nextBigInteger();

      // Input prime modulus p
      System.out.print("Enter the prime modulus p: ");
      BigInteger p = scanner.nextBigInteger();

      // Input generator g
      System.out.print("Enter the hash h: ");
      BigInteger h = scanner.nextBigInteger();
      BigInteger exp = p.subtract(BigInteger.ONE).divide(q);

      // Calculate g modulo p
      BigInteger g = h.modPow(exp, p);
      System.out.print("g :" +g+"\n");
      // System.out.print("Enter the generator g: ");
      // BigInteger g = h;

      // Input private key x
      System.out.print("Enter the private key x (0 < x < q): ");
      BigInteger x = scanner.nextBigInteger();

      // Calculate public key y
      BigInteger y = g.modPow(x, p);

      // Display public and private keys
      System.out.println("Public key {p, q, g, y}: {" + p + ", " + q + ", " + g + ", " + y + "}");
      System.out.println("Private key {p, q, g, x}: {" + p + ", " + q + ", " + g + ", " + x + "}");

      // Input message hash value h
      System.out.print("Enter the hash value h: ");
      BigInteger H = scanner.nextBigInteger();

      // Input random number k
      System.out.print("Enter the random number k (0 < k < q): ");
```

```
        BigInteger k = scanner.nextBigInteger();

        // Calculate r
        BigInteger r = g.modPow(k, p).mod(q);

        // Calculate inverse of k mod q
        BigInteger i = k.modInverse(q);

        // Calculate s
        BigInteger s = i.multiply(H.add(x.multiply(r))).mod(q);

        // Display digital signature {r, s}
        System.out.println("Digital signature {r, s}: {" + r + ", " + s + "}");

        // Input signature to verify
        System.out.print("Enter the signature to verify (r s): ");
        BigInteger vr = scanner.nextBigInteger();
        BigInteger vs = scanner.nextBigInteger();

        // Calculate w
        BigInteger w = vs.modInverse(q);

        // Calculate u1 and u2
        BigInteger u1 = h.multiply(w).mod(q);
        BigInteger u2 = r.multiply(w).mod(q);

        // Calculate v
        BigInteger v = g.modPow(u1, p).multiply(y.modPow(u2, p)).mod(p).mod(q);

        // Check if v matches the received r
        boolean verified = v.equals(vr);
        if (verified) {
            System.out.println("Verification passed.");
        } else {
            System.out.println("Verification failed.");
        }

        scanner.close();
    }
}
```

EXPERIMENT-12

Objective :- Implement DES Encryption.

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
import java.util.Base64;
import java.util.Scanner;

public class DESEncryption {

    public static String encrypt(String data, String key) {
        try {
            DESKeySpec desKeySpec = new DESKeySpec(key.getBytes());
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
            SecretKey secretKey = keyFactory.generateSecret(desKeySpec);
            Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            byte[] encryptedBytes = cipher.doFinal(data.getBytes());
            return Base64.getEncoder().encodeToString(encryptedBytes);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    public static String decrypt(String encryptedData, String key) {
        try {
            DESKeySpec desKeySpec = new DESKeySpec(key.getBytes());
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
            SecretKey secretKey = keyFactory.generateSecret(desKeySpec);
            Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] decodedBytes = Base64.getDecoder().decode(encryptedData);
            byte[] decryptedBytes = cipher.doFinal(decodedBytes);
            return new String(decryptedBytes);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

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

        System.out.println("Enter the data to encrypt:");
        String originalData = scanner.nextLine();
```

```
        System.out.println("Enter the encryption key (8 characters):");
        String key = scanner.nextLine();

        String encryptedData = encrypt(originalData, key);
        System.out.println("Encrypted Data: " + encryptedData);

        String decryptedData = decrypt(encryptedData, key);
        System.out.println("Decrypted Data: " + decryptedData);

        scanner.close();
    }
}
```