



**GLA**  
**UNIVERSITY**  
**MATHURA**  
Established vide U.P. Act 21 of 2010.

**DEPARTMENT OF COMPUTER ENGINEERING & APPLICATIONS**

**Institute of Engineering & Technology**

**Cryptography & Network Security Lab (BCSE-0071)**

**Name: Yash Gupta**

**Section: G (70)**

**University Roll No.: 201500817**

**Course: B.Tech - CSE**

**Submitted to: Mr.Ashish Srivastava**

# INDEX

Serial No.	Title	Sign.
1.	Write a program to implement Additive Cipher (Z26) with the following conditions: Plaintext should be in lowercase, Cipher text should be uppercase, Brute force attack	
2.	Write a program to implement Multiplicative Cipher: Plaintext should be in lowercase, Cipher text should be uppercase, Brute force attack.	
3.	Write a program to implement Affine Cipher: Plaintext should be in lowercase, Cipher text should be uppercase, Brute force attack.	
4.	Write a program in to implement Auto key Cipher: Plaintext should be in lowercase, Cipher text should be uppercase, Brute force attack.	
5.	Write a program to implement Play fair Cipher to encrypt & decrypt the given message where the key matrix can be formed by using a given keyword.	
6.	Write a program to implement Hill Cipher to encrypt & decrypt the given message by using a given key matrix. Show the values for key and its corresponding key inverse values.	
7.	Write a program to implement Elgamal Cryptosystem to generate the pair of keys and then show the encryption & decryption of a given message.	
8.	Write a program to implement Rabin Miller Primality Test to check whether given number is prime or composite.	
9.	Write a program to implement Diffie-Hellman key exchange Algorithm to exchange the symmetric key and show the encryption & decryption.	
10.	Write a program to implement RSA Algorithm to generate a pair of keys and show the encryption and decryption by using a given key pair.	
11.	Write a program to implement Elgamal algorithm for implementing digital signature.	

## Experiment 1

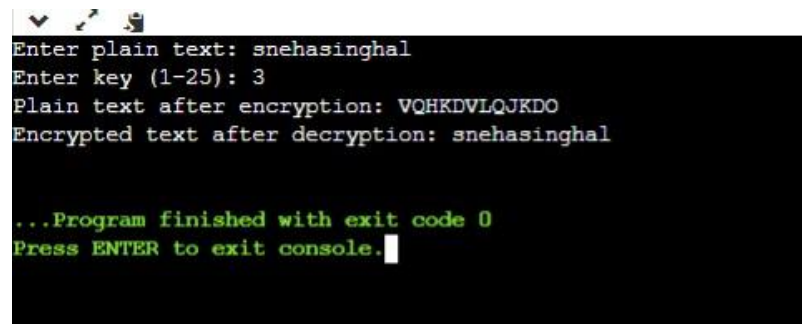
Write a program to implement Additive Cipher ( $Z_{26}$ ) with the following conditions:

- Plaintext should be in lowercase.
- Ciphertext should be uppercase.
- Brute force attack.

**Source Code:**

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter plain text: ");
        String plaintext = sc.nextLine();
        System.out.print("Enter key (1-25): ");
        int key = sc.nextInt();
        String encryptText = encrypt(plaintext, key);
        System.out.print("Plain text after encryption: ");
        System.out.println(encryptText);
        String decryptText = decrypt(encryptText, key);
        System.out.print("Encrypted text after decryption: ");
        System.out.println(decryptText);
    }
    public static String encrypt(String plaintext, int key) {
        StringBuilder encryptText = new StringBuilder();
        for (int i = 0; i < plaintext.length(); i++) {
            char ch = plaintext.charAt(i);
            if (Character.isLowerCase(ch)) {
                char shiftedChar = (char) ((ch - 'a' + key) % 26 + 'A');
                encryptText.append(shiftedChar);
            }
        }
        return encryptText.toString();
    }
    public static String decrypt(String encryptText, int key) {
        StringBuilder decryptText = new StringBuilder();
        for (int i = 0; i < encryptText.length(); i++) {
            char ch = encryptText.charAt(i);
            if (Character.isUpperCase(ch)) {
                char shiftedChar = (char) ((ch - 'A' - key + 26) % 26 + 'a');
                decryptText.append(shiftedChar);
            }
        }
        return decryptText.toString();
    }
}
```

**Input & Output:**



```
Enter plain text: snehasinghal
Enter key (1-25): 3
Plain text after encryption: VQHKDVLQJKDO
Encrypted text after decryption: snehasinghal

...Program finished with exit code 0
Press ENTER to exit console.
```

## Experiment 2

Write a program to implement Multiplicative Cipher.

- Plaintext should be in lowercase.
- Ciphertext should be uppercase.
- Brute force attack.

Source Code:

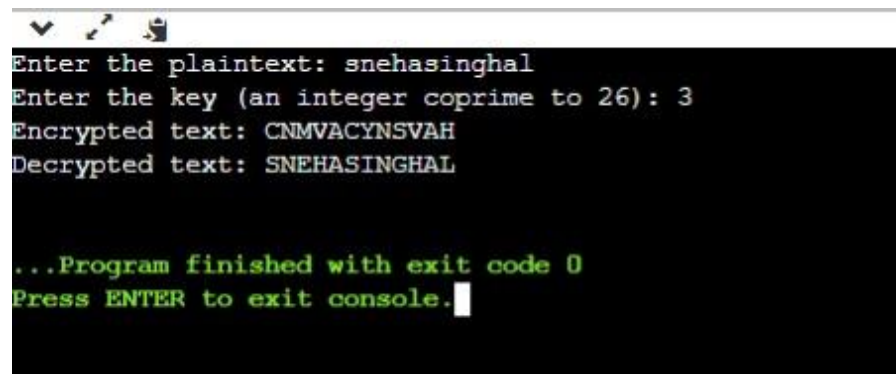
```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the plaintext: ");
        String plaintext = scanner.nextLine().toUpperCase();
        System.out.print("Enter the key (an integer coprime to 26): ");
        int key = scanner.nextInt();
        if (!isCoprime(key, 26)) {
            System.out.println("Invalid key. Key must be an integer coprime to 26.");
            return;
        }
        String ciphertext = encrypt(plaintext, key);
        System.out.println("Encrypted text: " + ciphertext);
        String decryptedText = decrypt(ciphertext, key);
        System.out.println("Decrypted text: " + decryptedText);
    }
    private static String encrypt(String plaintext, int key) {
        StringBuilder ciphertext = new StringBuilder();
        for (char ch : plaintext.toCharArray()) {
            if (ch == ' ') {
                ciphertext.append(ch);
            } else {
                int charValue = ch - 'A';
                int encryptedValue = (charValue * key) % 26;
                ciphertext.append((char) (encryptedValue + 'A'));
            }
        }
        return ciphertext.toString();
    }
    private static String decrypt(String ciphertext, int key) {
        StringBuilder decryptedText = new StringBuilder();
        int modInverse = modInverse(key, 26);
        for (char ch : ciphertext.toCharArray()) {
            if (ch == ' ') {
                decryptedText.append(ch);
            } else {
                int charValue = ch - 'A';
                int decryptedValue = (charValue * modInverse) % 26;
                decryptedText.append((char) (decryptedValue + 'A'));
            }
        }
        return decryptedText.toString();
    }
    private static boolean isCoprime(int a, int b) {
        return gcd(a, b) == 1;
    }
    private static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }
}
```

```

        int decryptedValue = (charValue * modInverse) % 26;
        if (decryptedValue < 0) {
            decryptedValue += 26; // Handle negative values
        }
        decryptedText.append((char) (decryptedValue + 'A'));
    }
}
return decryptedText.toString();
}
private static boolean isCoprime(int a, int b) {
    return gcd(a, b) == 1;
}
private static int gcd(int a, int b) {
    if (b == 0) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}
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;
}
}
}

```

### Input & Output:



```

Enter the plaintext: snehasinghal
Enter the key (an integer coprime to 26): 3
Encrypted text: CNMVACYNSVAH
Decrypted text: SNEHASINGHAL

...Program finished with exit code 0
Press ENTER to exit console.

```

### Experiment 3

Write a program to implement Affine Cipher.

- Plaintext should be in lowercase.
- Ciphertext should be uppercase.
- Brute force attack.

#### Source Code:

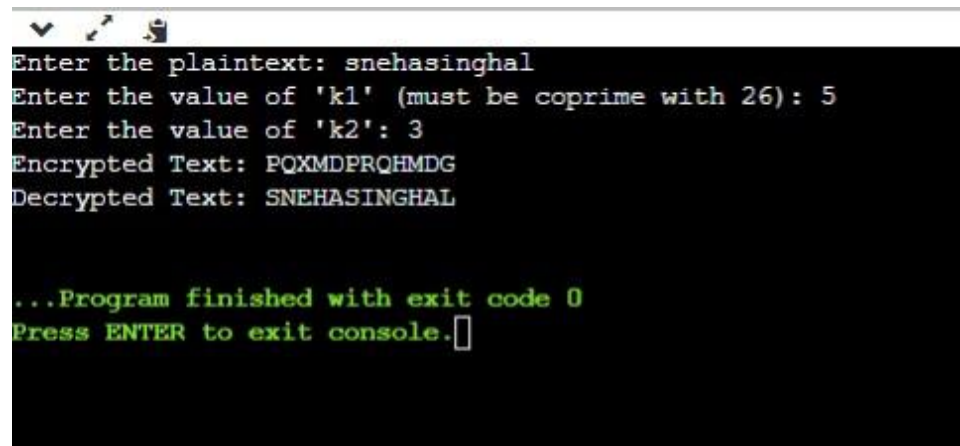
```
import java.util.Scanner;
public class Main {
    private static int modInverse(int k1, int m) {
        for (int i = 1; i < m; i++)
            if ((k1 * i) % m == 1)
                return i;
        return -1;
    }
    private static String encrypt(String plaintext, int k1, int k2) {
        StringBuilder ciphertext = new StringBuilder();
        for (char ch : plaintext.toCharArray()) {
            if (Character.isLetter(ch)) {
                int x = (Character.toUpperCase(ch) - 'A');
                int encryptedChar = (k1 * x + k2) % 26;
                ciphertext.append((char) (encryptedChar + 'A'));
            } else {
                ciphertext.append(ch);
            }
        }
        return ciphertext.toString();
    }
    private static String decrypt(String ciphertext, int k1, int k2) {
        StringBuilder decryptedText = new StringBuilder();
        int aInverse = modInverse(k1, 26);
        if (aInverse == -1) {
            System.out.println("Invalid key. The key must be chosen such that 'k1' and 'm' are coprime.");
            return "";
        }
        for (char ch : ciphertext.toCharArray()) {
            if (Character.isLetter(ch)) {
                int y = (Character.toUpperCase(ch) - 'A');
                int decryptedChar = (aInverse * (y - k2 + 26)) % 26;
                decryptedText.append((char) (decryptedChar + 'A'));
            } else {
                decryptedText.append(ch);
            }
        }
        return decryptedText.toString();
    }
}
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the plaintext: ");
    String plaintext = scanner.nextLine().toUpperCase();
    System.out.print("Enter the value of 'k1' (must be coprime with 26): ");
    int k1 = scanner.nextInt();
    System.out.print("Enter the value of 'k2': ");
    int k2 = scanner.nextInt();
    if (k1 < 0 || k1 >= 26 || gcd(k1, 26) != 1) {
        System.out.println("Invalid value of 'k1'. It must be coprime with 26.");
        return;
    }
    String encryptedText = encrypt(plaintext, k1, k2);
    System.out.println("Encrypted Text: " + encryptedText);
    String decryptedText = decrypt(encryptedText, k1, k2);
    System.out.println("Decrypted Text: " + decryptedText);
}
private static int gcd(int k1, int k2) {
    if (k2 == 0)
        return k1;
    return gcd(k2, k1 % k2);
}
}

```

### Input & Output:



```

Enter the plaintext: snehasinghal
Enter the value of 'k1' (must be coprime with 26): 5
Enter the value of 'k2': 3
Encrypted Text: PQXMDPRQHMDG
Decrypted Text: SNEHASINGHAL

...Program finished with exit code 0
Press ENTER to exit console.

```

## Experiment 4

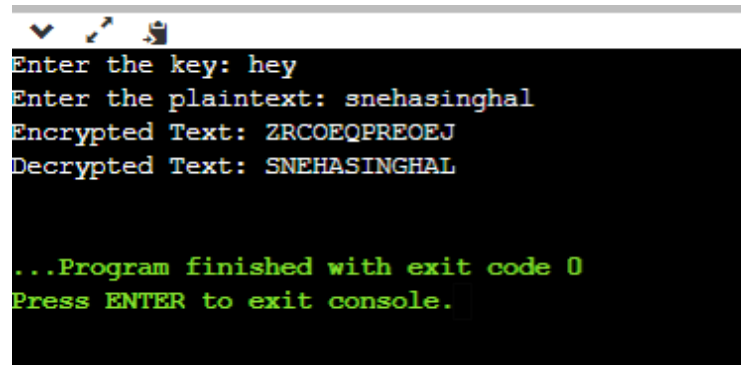
Write a program in to implement Autokey Cipher.

- Plaintext should be in lowercase.
- Ciphertext should be uppercase.
- Brute force attack.

**Source Code:**

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the key: ");
        String key = scanner.nextLine().toUpperCase();
        System.out.print("Enter the plaintext: ");
        String plaintext = scanner.nextLine().toUpperCase();
        String ciphertext = encrypt(plaintext, key);
        System.out.println("Encrypted Text: " + ciphertext);
        String decryptedText = decrypt(ciphertext, key);
        System.out.println("Decrypted Text: " + decryptedText);
    }
    private static String encrypt(String plaintext, String key) {
        StringBuilder ciphertext = new StringBuilder();
        int keyIndex = 0;
        for (char c : plaintext.toCharArray()) {
            if (Character.isLetter(c)) {
                char encryptedChar = (char) ((c + key.charAt(keyIndex) - 2 * 'A') % 26 + 'A');
                ciphertext.append(encryptedChar);
                keyIndex = (keyIndex + 1) % key.length();
            } else {
                ciphertext.append(c);
            }
        }
        return ciphertext.toString();
    }
    private static String decrypt(String ciphertext, String key) {
        StringBuilder decryptedText = new StringBuilder();
        int keyIndex = 0;
        for (char c : ciphertext.toCharArray()) {
            if (Character.isLetter(c)) {
                char decryptedChar = (char) ((c - key.charAt(keyIndex) + 26) % 26 + 'A');
                decryptedText.append(decryptedChar);
                keyIndex = (keyIndex + 1) % key.length();
            } else {
                decryptedText.append(c);
            }
        }
        return decryptedText.toString();
    }
}
```

**Input & Output:**



```
Enter the key: hey
Enter the plaintext: snehasinghal
Encrypted Text: ZRCOEQPREOEJ
Decrypted Text: SNEHASINGHAL

...Program finished with exit code 0
Press ENTER to exit console.
```



## Experiment 5

Write a program to implement Playfair Cipher to encrypt & decrypt the given message where the key matrix can be formed by using a given keyword.

### Source Code:

```
import java.util.Scanner;
public class Main {
    private static char[][] keyMatrix;
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the keyword: ");
        String keyword = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");
        System.out.print("Enter the message to encrypt: ");
        String message = scanner.nextLine().toUpperCase().replaceAll("[^A-Z]", "");
        generateKeyMatrix(keyword);
        System.out.println("Key Matrix:");
        printKeyMatrix();
        String encryptedMessage = encrypt(message);
        System.out.println("Encrypted Message: " + encryptedMessage);
        String decryptedMessage = decrypt(encryptedMessage);
        System.out.println("Decrypted Message: " + decryptedMessage);
    }
    private static void generateKeyMatrix(String keyword) {
        keyMatrix = new char[5][5];
        String key = keyword + "ABCDEFGHIKLMNOPQRSTUVWXYZ";
        key = key.replaceAll("J", "I");
        boolean[] used = new boolean[26];
        int row = 0, col = 0;
        for (char c : key.toCharArray()) {
            if (!used[c - 'A']) {
                keyMatrix[row][col] = c;
                used[c - 'A'] = true;
                col++;
                if (col == 5) {
                    col = 0;
                    row++;
                }
            }
        }
    }
}
```

```

private static void printKeyMatrix() {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            System.out.print(keyMatrix[i][j] + " ");
        }
        System.out.println();
    }
}

private static String formatMessage(String message) {
    message = message.replaceAll("J", "I");
    StringBuilder formattedMessage = new StringBuilder();
    for (int i = 0; i < message.length(); i += 2) {
        formattedMessage.append(message.charAt(i));
        if (i + 1 < message.length()) {
            if (message.charAt(i) == message.charAt(i + 1)) {
                formattedMessage.append('X');
                i--;
            } else {
                formattedMessage.append(message.charAt(i + 1));
            }
        } else {
            formattedMessage.append('X');
        }
    }
    return formattedMessage.toString();
}

private static String encrypt(String message) {
    message = formatMessage(message);
    StringBuilder encryptedMessage = new StringBuilder();
    for (int i = 0; i < message.length(); i += 2) {
        char c1 = message.charAt(i);
        char c2 = message.charAt(i + 1);
        int[] pos1 = findPosition(c1);
        int[] pos2 = findPosition(c2);
        int row1 = pos1[0];
        int col1 = pos1[1];
        int row2 = pos2[0];
        int col2 = pos2[1];
        if (row1 == row2) {

```

```

            encryptedMessage.append(keyMatrix[row1][(col1 + 1) % 5]);
            encryptedMessage.append(keyMatrix[row2][(col2 + 1) % 5]);
        } else if (col1 == col2) {
            encryptedMessage.append(keyMatrix[(row1 + 1) % 5][col1]);
            encryptedMessage.append(keyMatrix[(row2 + 1) % 5][col2]);
        } else {
            encryptedMessage.append(keyMatrix[row1][col2]);
            encryptedMessage.append(keyMatrix[row2][col1]);
        }
    }
    return encryptedMessage.toString();
}

private static String decrypt(String message) {
    StringBuilder decryptedMessage = new StringBuilder();
    for (int i = 0; i < message.length(); i += 2) {
        char c1 = message.charAt(i);
        char c2 = message.charAt(i + 1);
        int[] pos1 = findPosition(c1);
        int[] pos2 = findPosition(c2);
        int row1 = pos1[0];
        int col1 = pos1[1];
        int row2 = pos2[0];
        int col2 = pos2[1];
        if (row1 == row2) {
            decryptedMessage.append(keyMatrix[row1][(col1 + 4) % 5]);
            decryptedMessage.append(keyMatrix[row2][(col2 + 4) % 5]);
        } else if (col1 == col2) {
            decryptedMessage.append(keyMatrix[(row1 + 4) % 5][col1]);
            decryptedMessage.append(keyMatrix[(row2 + 4) % 5][col2]);
        } else {
            decryptedMessage.append(keyMatrix[row1][col2]);
            decryptedMessage.append(keyMatrix[row2][col1]);
        }
    }
    return decryptedMessage.toString();
}

```

```

private static int[] findPosition(char c) {
    int[] position = new int[2];
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (keyMatrix[i][j] == c) {
                position[0] = i;
                position[1] = j;
                return position;
            }
        }
    }
    return position;
}
}

```

### Input & Output:

```

Enter the keyword: keyword
Enter the message to encrypt: snehasinghal
Key Matrix:
K E Y W O
R D A B C
F G H I L
M N P Q S
T U V X Z
Encrypted Message: MPYGCPGQHICH
Decrypted Message: SNEHASINGHAL

...Program finished with exit code 0
Press ENTER to exit console.

```

## Experiment 6

Write a program to implement Hill Cipher to encrypt & decrypt the given message by using a given key matrix. Show the values for key and its corresponding key inverse values.

### Source Code:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Hill Cipher Implementation");
        System.out.println("Enter the key matrix size (n x n): ");
        int n = scanner.nextInt();
        int[][] keyMatrix = new int[n][n];
        System.out.println("Enter the key matrix elements:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                keyMatrix[i][j] = scanner.nextInt() % 26;
            }
        }
        int[][] keyInverse = getKeyInverse(keyMatrix, n);
        System.out.println("Key Matrix:");
        displayMatrix(keyMatrix);
        System.out.println("Key Inverse:");
        displayMatrix(keyInverse);
        System.out.println("Enter the plaintext (in uppercase):");
        scanner.nextLine();
        String input = scanner.nextLine();
        String ciphertext = encrypt(input, keyMatrix, n);
        System.out.println("Encrypted Text: " + ciphertext);
        String decryptedText = decrypt(ciphertext, keyInverse, n);
        System.out.println("Decrypted Text: " + decryptedText);
    }
    private static String encrypt(String plaintext, int[][] keyMatrix, int n) {
        StringBuilder ciphertext = new StringBuilder();
        while (plaintext.length() % n != 0) {
            plaintext += 'X';
        }
        for (int i = 0; i < plaintext.length(); i += n) {
            String block = plaintext.substring(i, i + n);
            for (int j = 0; j < n; j++) {
                int sum = 0;
                for (int k = 0; k < n; k++) {
                    sum += (keyMatrix[j][k] * (block.charAt(k) - 'A')) % 26;
                    sum %= 26;
                }
                ciphertext.append((char) ('A' + sum));
            }
        }
    }
}
```

```

        ciphertext.append((char) ('A' + sum));
    }
    return ciphertext.toString();
}
private static String decrypt(String ciphertext, int[][] keyInverse, int n) {
    StringBuilder plaintext = new StringBuilder();
    for (int i = 0; i < ciphertext.length(); i += n) {
        String block = ciphertext.substring(i, i + n);
        for (int j = 0; j < n; j++) {
            int sum = 0;
            for (int k = 0; k < n; k++) {
                sum += (keyInverse[j][k] * (block.charAt(k) - 'A' + 26)) % 26;
                sum %= 26;
            }
            plaintext.append((char) ('A' + sum));
        }
    }
    return plaintext.toString();
}
private static int[][] getKeyInverse(int[][] keyMatrix, int n) {
    int det = determinant(keyMatrix, n);
    int detInverse = modInverse(det, 26);
    int[][] adjugate = adjugate(keyMatrix, n);
    int[][] keyInverse = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            keyInverse[i][j] = (adjugate[i][j] * detInverse) % 26;
            if (keyInverse[i][j] < 0) {
                keyInverse[i][j] += 26;
            }
        }
    }
    return keyInverse;
}
private static int determinant(int[][] matrix, int n) {
    if (n == 1) {
        return matrix[0][0];
    }
    if (n == 2) {
        return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0] + 26) % 26;
    }
    return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0] + 26) % 26;
}

```

```

    return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0] + 26) % 26;
}
int det = 0;
for (int i = 0; i < n; i++) {
    int[][] submatrix = new int[n - 1][n - 1];
    for (int j = 1; j < n; j++) {
        for (int k = 0, l = 0; k < n; k++) {
            if (k != i) {
                submatrix[j - 1][l++] = matrix[j][k];
            }
        }
    }
    int sign = (i % 2 == 0) ? 1 : -1;
    det = (det + sign * matrix[0][i] * determinant(submatrix, n - 1) + 26) % 26;
}
return det;
}
private static int[][] adjugate(int[][] matrix, int n) {
    int[][] adjugate = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int[][] submatrix = new int[n - 1][n - 1];
            for (int k = 0, l = 0; k < n; k++) {
                if (k != i) {
                    for (int m = 0, n1 = 0; m < n; m++) {
                        if (m != j) {
                            submatrix[l][n1++] = matrix[k][m];
                        }
                    }
                    l++;
                }
            }
            int sign = ((i + j) % 2 == 0) ? 1 : -1;
            adjugate[j][i] = (sign * determinant(submatrix, n - 1) + 26) % 26;
        }
    }
    return adjugate;
}
}

```

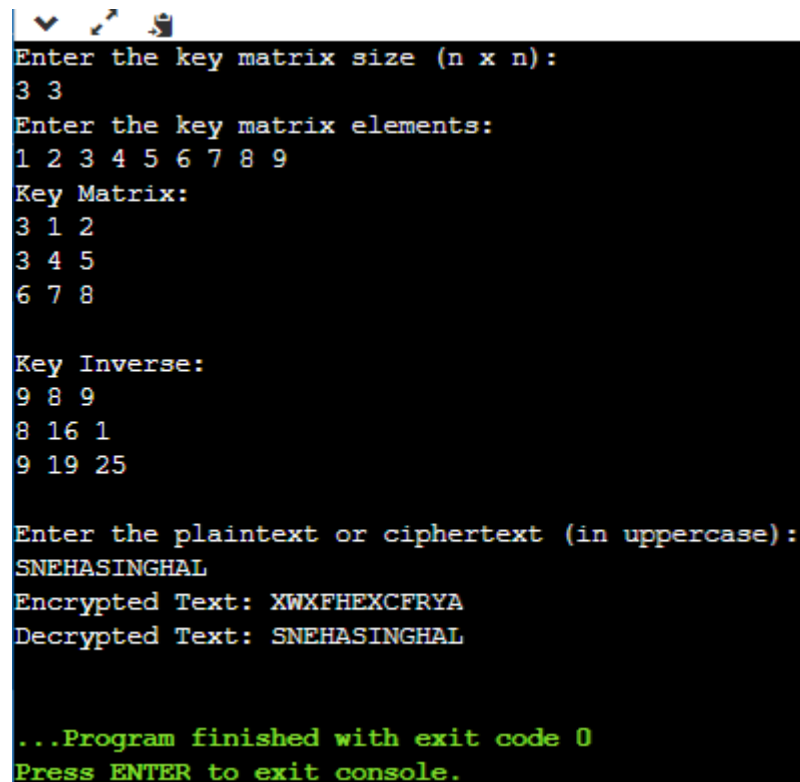
```

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;
}

private static void displayMatrix(int[][] matrix) {
    for (int[] row : matrix) {
        for (int element : row) {
            System.out.print(element + " ");
        }
        System.out.println();
    }
    System.out.println();
}
}

```

### Input & Output:



The screenshot shows a Java application running in a console. It prompts the user to enter the key matrix size (n x n), which is 3 3. Then it prompts for the key matrix elements, which are 1 2 3 4 5 6 7 8 9. The program then displays the key matrix as a 3x3 grid. Next, it prompts for the key inverse, which is 9 8 9, 8 16 1, 9 19 25. Then it prompts for the plaintext or ciphertext (in uppercase), which is SNEHASINGHAL. The program then displays the encrypted text as XW XFHEXCFRYA and the decrypted text as SNEHASINGHAL. Finally, it displays the message "...Program finished with exit code 0" and "Press ENTER to exit console."

```

Enter the key matrix size (n x n):
3 3
Enter the key matrix elements:
1 2 3 4 5 6 7 8 9
Key Matrix:
3 1 2
3 4 5
6 7 8

Key Inverse:
9 8 9
8 16 1
9 19 25

Enter the plaintext or ciphertext (in uppercase):
SNEHASINGHAL
Encrypted Text: XW XFHEXCFRYA
Decrypted Text: SNEHASINGHAL

...Program finished with exit code 0
Press ENTER to exit console.

```

## Experiment 7

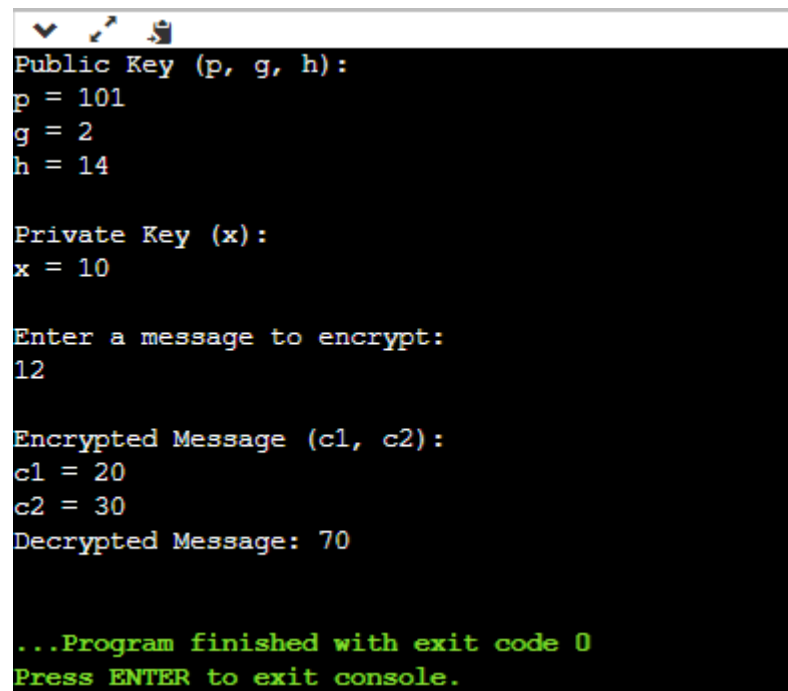
Write a program to implement Elgamal Cryptosystem to generate the pair of keys and then show the encryption & decryption of a given message.

**Source Code:**

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BigInteger p = generatePrime();
        BigInteger g = generatePrimitiveRoot(p);
        BigInteger x = generatePrivateKey(p);
        BigInteger h = g.modPow(x, p);
        System.out.println("Public Key (p, g, h):");
        System.out.println("p = " + p);
        System.out.println("g = " + g);
        System.out.println("h = " + h);
        System.out.println("\nPrivate Key (x):");
        System.out.println("x = " + x);
        System.out.println("\nEnter a message to encrypt:");
        String message = scanner.nextLine();
        BigInteger k = generateRandomK(p);
        BigInteger c1 = g.modPow(k, p);
        BigInteger c2 = h.modPow(k, p).multiply(messageToBigInteger(message)).mod(p);
        System.out.println("\nEncrypted Message (c1, c2):");
        System.out.println("c1 = " + c1);
        System.out.println("c2 = " + c2);
        BigInteger s = c1.modPow(x, p);
        BigInteger sInverse = s.modInverse(p);
        BigInteger decryptedMessage = c2.multiply(sInverse).mod(p);
        System.out.println("Decrypted Message: " + decryptedMessage);
    }
    private static BigInteger generatePrime() {
        return new BigInteger("101");
    }
}
```

```
private static BigInteger generatePrimitiveRoot(BigInteger p) {
    return new BigInteger("2");
}
private static BigInteger generatePrivateKey(BigInteger p) {
    return new BigInteger(p.bitLength() - 2, new SecureRandom()).add(BigInteger.valueOf(2));
}
private static BigInteger generateRandomK(BigInteger p) {
    return new BigInteger(p.bitLength() - 2, new SecureRandom()).add(BigInteger.valueOf(2));
}
private static BigInteger messageToBigInteger(String message) {
    byte[] bytes = message.getBytes();
    return new BigInteger(bytes);
}
}
```

### Input & Ouput:

A screenshot of a console window with a black background and white text. The window has a title bar with standard OS icons. The text displayed is the output of a program, showing the calculation of a public key, a private key, encryption of a message, and decryption of the result. The program ends with a green-colored message indicating successful completion.

```
Public Key (p, g, h):  
p = 101  
g = 2  
h = 14  
  
Private Key (x):  
x = 10  
  
Enter a message to encrypt:  
12  
  
Encrypted Message (c1, c2):  
c1 = 20  
c2 = 30  
Decrypted Message: 70  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```



## Experiment 8

Write a program to implement Rabin Miller Primality Test to check whether given number is prime or composite.

Source Code:

```
import java.util.Random;
import java.util.Scanner;
public class Main {
    private static long power(long a, long b, long m) {
        long result = 1;
        a = a % m;
        while (b > 0) {
            if (b % 2 == 1)
                result = (result * a) % m;
            b = b >> 1;
            a = (a * a) % m;
        }
        return result;
    }
    private static boolean isPrime(long n, int k) {
        if (n <= 1 || n == 4)
            return false;
        if (n <= 3)
            return true;
        long d = n - 1;
        while (d % 2 == 0)
            d /= 2;
        for (int i = 0; i < k; i++) {
            if (!millerTest(n, d))
                return false;
        }
        return true;
    }
    private static boolean millerTest(long n, long d) {
        Random rand = new Random();
        long a = 2 + rand.nextInt((int) (n - 4));
        long x = power(a, d, n);
        if (x == 1 || x == n - 1)
            return true;
    }
}
```

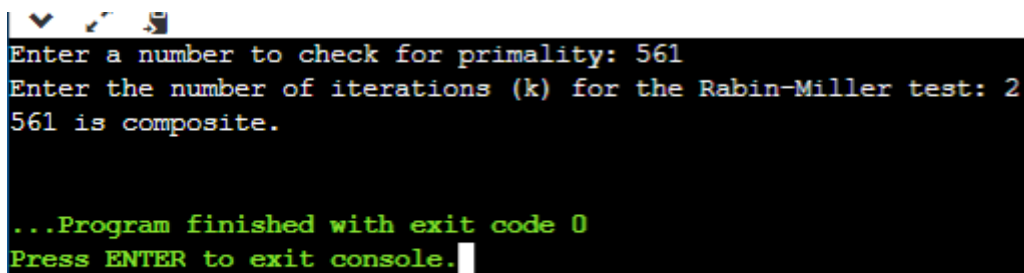
```

        while (d != n - 1) {
            x = (x * x) % n;
            d *= 2;
            if (x == 1)
                return false;
            if (x == n - 1)
                return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number to check for primality: ");
        long number = scanner.nextLong();
        System.out.print("Enter the number of iterations (k) for the Rabin-Miller test: ");
        int k = scanner.nextInt();
        if (isPrime(number, k)) {
            System.out.println(number + " is prime.");
        } else {
            System.out.println(number + " is composite.");
        }
        scanner.close();
    }
}

```

#### Input & Output:

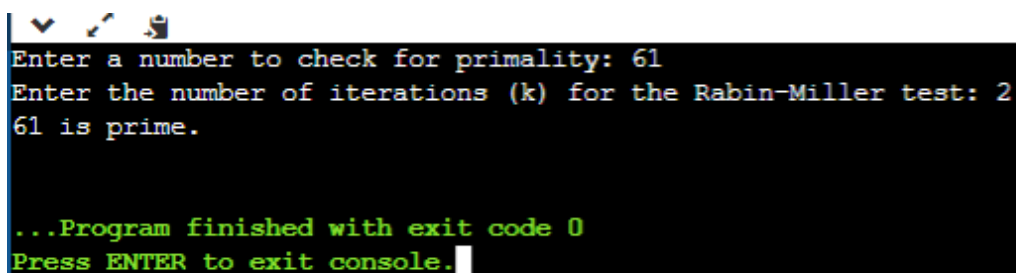


```

Enter a number to check for primality: 561
Enter the number of iterations (k) for the Rabin-Miller test: 2
561 is composite.

...Program finished with exit code 0
Press ENTER to exit console.

```



```

Enter a number to check for primality: 61
Enter the number of iterations (k) for the Rabin-Miller test: 2
61 is prime.

...Program finished with exit code 0
Press ENTER to exit console.

```

## Experiment 9

Write a program to implement Diffie-Hellman key exchange Algorithm to exchange the symmetric key and show the encryption & decryption.

### Source Code:

```
import java.math.BigInteger;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a prime number (p): ");
        BigInteger p = new BigInteger(scanner.nextLine());
        System.out.print("Enter a primitive root modulo p (g): ");
        BigInteger g = new BigInteger(scanner.nextLine());
        System.out.print("Enter private key for Alice (a): ");
        BigInteger a = new BigInteger(scanner.nextLine());
        System.out.print("Enter private key for Bob (b): ");
        BigInteger b = new BigInteger(scanner.nextLine());
        BigInteger A = g.modPow(a, p);
        BigInteger B = g.modPow(b, p);
        BigInteger secretKeyA = B.modPow(a, p);
        BigInteger secretKeyB = A.modPow(b, p);
        System.out.println("Shared Secret Key (Alice): " + secretKeyA);
        System.out.println("Shared Secret Key (Bob): " + secretKeyB);
        String message = "Sneha Singhal";
        System.out.println("Original Message: " + message);
        BigInteger symmetricKey = secretKeyA;
        String encryptedMessage = encrypt(message, symmetricKey);
        System.out.println("Encrypted Message: " + encryptedMessage);
        String decryptedMessage = decrypt(encryptedMessage, symmetricKey);
        System.out.println("Decrypted Message: " + decryptedMessage);
    }
    private static String encrypt(String message, BigInteger key) {
        StringBuilder encrypted = new StringBuilder();
        for (int i = 0; i < message.length(); i++) {
            char c = message.charAt(i);
            encrypted.append((char) (c ^ key.intValue()));
        }
        return encrypted.toString();
    }
    private static String decrypt(String encryptedMessage, BigInteger key) {
        return encrypt(encryptedMessage, key);
    }
}
```

### Input & Output:

```
Enter a prime number (p): 7
Enter a primitive root modulo p (g): 3
Enter private key for Alice (a): 2
Enter private key for Bob (b): 3
Shared Secret Key (Alice): 1
Shared Secret Key (Bob): 1
Original Message: Sneha Singhal
Encrypted Message: Rodi`!Rhofi`m
Decrypted Message: Sneha Singhal

...Program finished with exit code 0
Press ENTER to exit console.
```

## Experiment 10

Write a program to implement RSA Algorithm to generate a pair of keys and show the encryption and decryption by using a given key pair.

### Source Code:

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter I prime number (p): ");
        int p = sc.nextInt();
        System.out.print("Enter II prime number (q): ");
        int q = sc.nextInt();
        int n = p * q;
        int phiN = (p - 1) * (q - 1);
        int e = 0;
        ArrayList<Integer> list = new ArrayList<Integer>();
        ArrayList<Integer> dlist = new ArrayList<Integer>();
        for (int i = 2; i < phiN; i++) {
            if (findGCD(i, phiN) == 1) {
                e = i;
                list.add(e);
            }
        }
        for (int i = 1; i < phiN; i++) {
            for (int j = 0; j < list.size(); j++) {
                if ((i * list.get(j)) % phiN == 1) {
                    dlist.add(i);
                    break;
                }
            }
        }
        for (int i = 0; i < list.size(); i++) {
            System.out.println("Public key: " + list.get(i) + " " + n);
        }
        System.out.println();
        for (int i = 0; i < dlist.size(); i++) {
            System.out.println("Private key: " + dlist.get(i) + " " + n);
        }
    }
}
```

```

        int plaintext = 42;
        int ciphertext = encrypt(plaintext, list.get(0), n);
        System.out.println("Encrypted: " + ciphertext);
        int decryptedText = decrypt(ciphertext, dList.get(0), n);
        System.out.println("Decrypted: " + decryptedText);
    }
    public static int encrypt(int message, int publicKey, int n) {
        return (int) (Math.pow(message, publicKey) % n);
    }
    public static int decrypt(int ciphertext, int privateKey, int n) {
        return (int) (Math.pow(ciphertext, privateKey) % n);
    }
    public static int findGCD(int e, int phiN) {
        while (phiN != 0) {
            if (e > phiN) {
                e = e - phiN;
            } else {
                phiN = phiN - e;
            }
        }
        return e;
    }
}

```

#### Input & Output:

```

Enter I prime number (p): 3
Enter II prime number (q): 5
Public key: 3 15
Public key: 5 15
Public key: 7 15

Private key: 3 15
Private key: 5 15
Private key: 7 15
Encrypted: 3
Decrypted: 12

...Program finished with exit code 0
Press ENTER to exit console.

```

## Experiment 11

Write a program to implement Elgamal algorithm for implementing digital signature.

### Source Code:

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a prime number (p):");
        BigInteger p = scanner.nextBigInteger();
        System.out.println("Enter a primitive root modulo p (g):");
        BigInteger g = scanner.nextBigInteger();
        System.out.println("Enter the private key (a):");
        BigInteger privateKey = scanner.nextBigInteger();
        BigInteger publicKey = g.modPow(privateKey, p);
        System.out.println("Public Key (A): " + publicKey);
        System.out.println("Enter the message to be signed:");
        String message = scanner.next();
        BigInteger k = generateRandomK(p.subtract(BigInteger.ONE));
        BigInteger r = g.modPow(k, p);
        BigInteger hashMessage = new BigInteger(HashFunction.hash(message.getBytes()));
        BigInteger inverseK = k.modInverse(p.subtract(BigInteger.ONE));
        BigInteger s = inverseK.multiply(hashMessage.subtract(privateKey.multiply(r))).mod(p.subtract(BigInteger.ONE));
        System.out.println("Signature (r, s): (" + r + ", " + s + ")");
        BigInteger v1 = g.modPow(hashMessage, p);
        BigInteger v2 = publicKey.modPow(r, p).multiply(r.modPow(s, p)).mod(p);
        if (v1.equals(v2)) {
            System.out.println("Signature is valid.");
        } else {
            System.out.println("Signature is invalid.");
        }
    }
}
```

```
private static BigInteger generateRandomK(BigInteger max) {
    SecureRandom random = new SecureRandom();
    BigInteger k;
    do {
        k = new BigInteger(max.bitLength(), random);
    } while (k.compareTo(BigInteger.ONE) <= 0 || k.compareTo(max) >= 0 || !k.gcd(max).equals(BigInteger.ONE));
    return k;
}

class HashFunction {
    public static byte[] hash(byte[] input) {
        try {
            java.security.MessageDigest digest = java.security.MessageDigest.getInstance("SHA-256");
            return digest.digest(input);
        } catch (java.security.NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

### Input & Output:

```
Enter a prime number (p):
5
Enter a primitive root modulo p (g):
3
Enter the private key (a):
7
Public Key (A): 2
Enter the message to be signed:
snehasinghal
Signature (r, s): (2, 1)
Signature is valid.
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```