CNS LAB EXPERIMENTS

1. Write a C program that contains a string (char pointer) with a value ‘Hello world’. The program should XOR each character in this string with 0 and display the result.

PROGRAM:

```
#include <stdio.h>


int main() {
   char *str = "Hello world";
   int i = 0;


   printf("Original String: %s\n", str);
   printf("After XOR with 0: ");


   while (str[i] != '\0') {
      char result = str[i] ^ 0; // XOR with 0
      printf("%c", result);
      i++;
   }


   return 0;
}
```

OUTPUT:

```
Original String: Hello world
After XOR with 0: Hello world
```

2. Write a C program that contains a string (char pointer) with a value ‘Hello world’. The program should AND or and XOR each character in this string with 127 and display the result.

PROGRAM:

```
#include <stdio.h>


int main() {
   char *str = "Hello world";
   int i;


   printf("Original String: %s\n", str);


   // AND with 127
   printf("After AND with 127: ");
   for (i = 0; str[i] != '\0'; i++) {
      char result = str[i] & 127;
      printf("%c", result);
   }
   printf("\n");


   // OR with 127
   printf("After OR with 127: ");
   for (i = 0; str[i] != '\0'; i++) {
      char result = str[i] | 127;
      printf("%c", result);
   }
   printf("\n");


   // XOR with 127
   printf("After XOR with 127: ");
   for (i = 0; str[i] != '\0'; i++) {
      char result = str[i] ^ 127;
```

```
        printf("%c", result);
    }
    printf("\n");

    return 0;
}
```

OUTPUT:

```
Original String: Hello world
After AND with 127: Hello world
After OR with 127:
After XOR with 127: 7□□□□_□□□□
```

3. Write a Java program to perform encryption and decryption using the following algorithms

a. Ceaser cipher

PROGRAM:

```
import java.io.*;

public class CaesarCipher {

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

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


      System.out.print("Enter any String: ");

      String str = br.readLine();


      System.out.print("Enter the Key: ");

      int key = Integer.parseInt(br.readLine());


      String encrypted = encrypt(str, key);

      System.out.println("\nEncrypted String: " + encrypted);


      String decrypted = decrypt(encrypted, key);

      System.out.println("Decrypted String: " + decrypted);

   }


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

      StringBuilder result = new StringBuilder();


      for (char ch : text.toCharArray()) {

         if (Character.isUpperCase(ch)) {

            result.append((char) ((ch - 'A' + key) % 26 + 'A'));

         } else if (Character.isLowerCase(ch)) {

            result.append((char) ((ch - 'a' + key) % 26 + 'a'));

         } else {

            result.append(ch); // keep spaces/symbols unchanged
```

```
            }
        }
        return result.toString();
    }

    public static String decrypt(String text, int key) {
        return encrypt(text, 26 - (key % 26)); // Caesar Cipher is symmetric
    }
}
```

OUTPUT:

```
Enter any String: king
Enter the Key: 2

Encrypted String: mkpi
Decrypted String: king
```

b. Substitution cipher

PROGRAM:

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

public class SubstitutionCipher {
   static Scanner sc = new Scanner(System.in);
   static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

   public static void main(String[] args) throws IOException {
      String a = "abcdefghijklmnopqrstuvwxyz";
      String b = "zyxwvutsrqponmlkjihgfedcba";

      System.out.print("Enter any string: ");
      String str = br.readLine();
      String decrypt = "";
      char c;

      for (int i = 0; i < str.length(); i++) {
         c = str.charAt(i);

         // Convert to lowercase for processing
         char lowerC = Character.toLowerCase(c);
         int j = a.indexOf(lowerC);

         if (j != -1) { // If it's a letter
            char enc = b.charAt(j);
            // Preserve case
            if (Character.isUpperCase(c)) {
               enc = Character.toUpperCase(enc);
            }
```

```
            decrypt = decrypt + enc;
          } else {
            // Keep non-alphabetic characters unchanged
            decrypt = decrypt + c;
          }
        }
        System.out.println("The encrypted data is: " + decrypt);
    }
}
```

OUTPUT:

Enter any string: KING

The encrypted data is: PRMT

5.Write a C/JAVA program to implement the Blowfish algorithm logic.

PROGRAM:

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

public class Blowfish {
    public static void main(String[] args) throws Exception {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter text to encrypt: ");
        String plaintext = sc.nextLine();

        // Generate a Blowfish key
        KeyGenerator keyGen = KeyGenerator.getInstance("Blowfish");
        keyGen.init(128); // key size
        SecretKey secretKey = keyGen.generateKey();

        // Encrypt
        Cipher cipher = Cipher.getInstance("Blowfish");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes());
        String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);

        // Decrypt
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        String decryptedText = new String(decryptedBytes);
```

```
        // Output
        System.out.println("\nPlaintext: " + plaintext);
        System.out.println("Encrypted (Base64): " + encryptedText);
        System.out.println("Decrypted: " + decryptedText);
    }
}
```

OUTPUT:

```
Plaintext: KING
Encrypted (Base64): OS3UvO94c4Y=
Decrypted: KING
```

6. Write a C/JAVA program to implement the Rijndael algorithm logic.

PROGRAM:

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

public class Rijndael {
   public static void main(String[] args) {
      try (Scanner sc = new Scanner(System.in)) {
         // Input plaintext and key
         System.out.print("Enter the plaintext: ");
         String plainText = sc.nextLine();

         System.out.print("Enter a 16-character key (128-bit): ");
         String key = sc.nextLine();

         // Validate key length
         if (key.length() != 16) {
            System.out.println("Key must be exactly 16 characters (128 bits)!");
            return;
         }

         // Encrypt the text
         SecretKeySpec secretKey = new SecretKeySpec(key.getBytes(), "AES");
         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

         cipher.init(Cipher.ENCRYPT_MODE, secretKey);
         byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
         String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);
```

```
            System.out.println("\nEncrypted Text (Base64): " + encryptedText);

            // Decrypt the text
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
            String decryptedText = new String(decryptedBytes);

            System.out.println("Decrypted Text: " + decryptedText);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
Enter the plaintext: KING
Enter a 16-character key (128-bit): ABCDEFGHIJKLMNOP

Encrypted Text (Base64): jgJ38Anzt/MmOhDg9zjvzg==
Decrypted Text: KING
```

8. Write a Java program to implement the RSA algorithm.

PROGRAM:

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

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

      // Step 1: Input two prime numbers
      System.out.print("Enter first prime number (p): ");
      BigInteger p = new BigInteger(sc.next());
      System.out.print("Enter second prime number (q): ");
      BigInteger q = new BigInteger(sc.next());

      // Step 2: Compute n = p * q
      BigInteger n = p.multiply(q);

      // Step 3: Compute phi = (p-1)*(q-1)
      BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

      // Step 4: Choose public key exponent e
      BigInteger e;
      do {
         e = new BigInteger(16, rand); // small random number
      } while ((e.compareTo(BigInteger.ONE) <= 0) ||
            (e.compareTo(phi) >= 0) ||
            !e.gcd(phi).equals(BigInteger.ONE));
```

```
        // Step 5: Compute private key exponent d
        BigInteger d = e.modInverse(phi);

        // Display public and private keys
        System.out.println("\nPublic key (n, e): (" + n + ", " + e + ")");
        System.out.println("Private key (n, d): (" + n + ", " + d + ")");

        // Step 6: Input message to encrypt
        System.out.print("\nEnter the message (integer) to encrypt: ");
        BigInteger msg = new BigInteger(sc.next());

        // Step 7: Encryption -> c = m^e mod n
        BigInteger cipher = msg.modPow(e, n);
        System.out.println("Encrypted message: " + cipher);

        // Step 8: Decryption -> m = c^d mod n
        BigInteger decrypted = cipher.modPow(d, n);
        System.out.println("Decrypted message: " + decrypted);
    }
}
```

OUTPUT:

```
Enter first prime number (p): 11
Enter second prime number (q): 13

Public key (n, e): (143, 31)
Private key (n, d): (143, 31)

Enter the message (integer) to encrypt: 8
Encrypted message: 96
Decrypted message: 8
```

9. Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript.

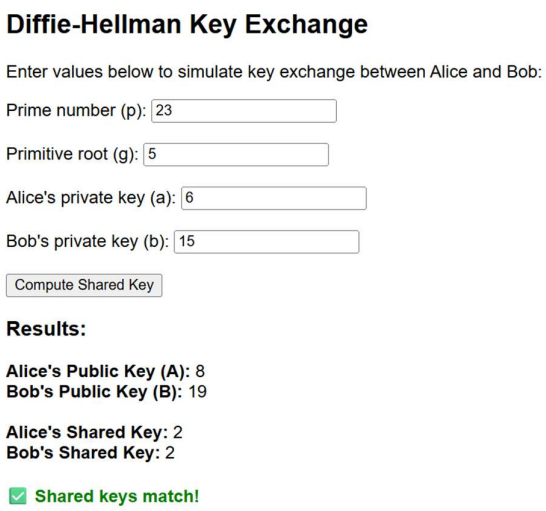
PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Diffie-Hellman Key Exchange</title>
</head>
<body style="font-family: Arial; margin: 40px;">
  <h2>Diffie-Hellman Key Exchange</h2>

  <p>Enter values below to simulate key exchange between Alice and Bob:</p>

  <label>Prime number (p): </label>
  <input type="number" id="p" value="23"><br><br>

  <label>Primitive root (g): </label>
  <input type="number" id="g" value="5"><br><br>

  <label>Alice's private key (a): </label>
  <input type="number" id="a" value="6"><br><br>

  <label>Bob's private key (b): </label>
  <input type="number" id="b" value="15"><br><br>

  <button onclick="computeDH()">Compute Shared Key</button>

  <h3>Results:</h3>
  <p id="output"></p>

  <script>
    function modExp(base, exp, mod) {
```

```
  let result = 1;
  base = base % mod;
  while (exp > 0) {
    if (exp % 2 === 1) result = (result * base) % mod;
    exp = Math.floor(exp / 2);
    base = (base * base) % mod;
  }
  return result;
}

function computeDH() {
  let p = parseInt(document.getElementById("p").value);
  let g = parseInt(document.getElementById("g").value);
  let a = parseInt(document.getElementById("a").value);
  let b = parseInt(document.getElementById("b").value);

  // Step 1: Compute public keys
  let A = modExp(g, a, p); // Alice's public key
  let B = modExp(g, b, p); // Bob's public key

  // Step 2: Compute shared secret keys
  let keyA = modExp(B, a, p); // Alice computes
  let keyB = modExp(A, b, p); // Bob computes

  document.getElementById("output").innerHTML =
    `<b>Alice's Public Key (A):</b> ${A}<br>` +
    `<b>Bob's Public Key (B):</b> ${B}<br><br>` +
    `<b>Alice's Shared Key:</b> ${keyA}<br>` +
    `<b>Bob's Shared Key:</b> ${keyB}<br><br>` +
    (keyA === keyB
      ? "<b style='color:green;'>✅ Shared keys match!</b>"
```

```
        : "<b style='color:red;'>❌ Keys do not match.</b>");
    }
  </script>
</body>
</html>
```

OUTPUT:



10. Calculate the message digest of a text using the SHA-1 algorithm in JAVA.

PROGRAM:

```
import java.security.MessageDigest;
import java.util.Scanner;

public class SHA1Digest {
   public static void main(String[] args) {
      try {
         // Create a Scanner object for user input
         Scanner sc = new Scanner(System.in);

         // Read input text from user
         System.out.print("Enter text to find its SHA-1 digest: ");
         String input = sc.nextLine();

         // Create MessageDigest instance for SHA-1
         MessageDigest md = MessageDigest.getInstance("SHA-1");

         // Update the MessageDigest object with input bytes
         md.update(input.getBytes());

         // Generate the message digest
         byte[] digest = md.digest();

         // Convert the byte array into hexadecimal format
         StringBuilder hexString = new StringBuilder();
         for (byte b : digest) {
            hexString.append(String.format("%02x", b & 0xff));
         }

         // Display the result
```

```
            System.out.println("SHA-1 Message Digest: " + hexString.toString());


            sc.close();
        } catch (Exception e) {
            System.out.println("Error while computing SHA-1: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
Enter text to find its SHA-1 digest: KING
SHA-1 Message Digest: 50b8a339f82ab9ce6c55bf8ea10dad8513e9d142
```

11. Calculate the message digest of a text using the MD5 algorithm in JAVA

PROGRAM:

```
import java.security.MessageDigest;
import java.util.Scanner;

public class MD5Digest {
   public static void main(String[] args) {
      try {
         // Create Scanner object for input
         Scanner sc = new Scanner(System.in);

         // Read text from user
         System.out.print("Enter text to find its MD5 digest: ");
         String input = sc.nextLine();

         // Create MessageDigest instance for MD5
         MessageDigest md = MessageDigest.getInstance("MD5");

         // Update the digest with input bytes
         md.update(input.getBytes());

         // Compute the message digest
         byte[] digest = md.digest();

         // Convert byte array into hexadecimal format
         StringBuilder hexString = new StringBuilder();
         for (byte b : digest) {
            hexString.append(String.format("%02x", b & 0xff));
         }

         // Display the result
```

```
            System.out.println("MD5 Message Digest: " + hexString.toString());


            sc.close();
        } catch (Exception e) {
            System.out.println("Error while computing MD5: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
Enter text to find its MD5 digest: KING
MD5 Message Digest: 123b28961dd0f97d91bcb55d7a3b7c1c
```