

## **EXPERIMENT: 1**

### **AIM:**

**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 displays the result.**

### **DESCRIPTION:**

XOR Encryption is an encryption method used to encrypt data and is hard to crack by brute-force method, i.e generating random encryption keys to match with the correct one.

The concept of implementation is to first define XOR – encryption key and then to perform XOR operation of the characters in the String with this key which you want to encrypt. To decrypt the encrypted characters we have to perform XOR operation again with the defined key. Here we are encrypting the entire String.

### **PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
main()
{
char str[]="Hello World";
char str1[11];
int i,len;
len=strlen(str);
for(i=0;i<len;i++)
{
str1[i]=str[i]^0;
printf("%c",str1[i]);
}
printf("\n");
}
```

### **OUTPUT:**

Hello World

## **EXPERIMENT: 2**

### **AIM:**

**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.**

### **DESCRIPTION:**

#### **AND Operation:**

There are two inputs and one output in binary **AND** operation. The inputs and result to binary **AND** operation can only be **0** or **1**. The binary **AND** operation will always produce a **1** output if both inputs are **1** and will produce a **0** output if both inputs are **0**. For two different inputs, the output will be **0**.

#### **XOR OPERATION:**

There are two inputs and one output in binary **XOR** (exclusive **OR**) operation. It is similar to **ADD** operation which takes two inputs and produces one result i.e. one output.

The inputs and result to a binary **XOR** operation can only be **0** or **1**. The binary **XOR** operation will always produce a **1** output if either of its is **1** and will produce a **0** output if both of its inputs are **0** or **1**.

### **PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int main()
{
    char str[]="Hello World";
    char str1[11];
    char str2[11],str3[11];
    int i,len;
    len = strlen(str);
    for(i=0;i<=11;i++)
        str2[i]=str[i];
    printf("%s\n",str);
    for(i=0;i<len;i++)
    {
        str1[i] = str[i]&127;
        printf("%c",str1[i]);
    }
```

```
printf("\n");  
for(i=0;i<len;i++)  
{  
    str3[i] = str2[i]^127;  
    printf("%c",str3[i]^127);  
}  
printf("\n");  
return 0;  
}
```

### **OUTPUT:**

Hello World

Hello World

Hello World

## **EXPERIMENT: 3**

### **AIM:**

**Write a java program to perform encryption and decryption using the following algorithms.**

### **DESCRIPTION:**

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus to cipher a given text we need an integer value, known as a shift which indicates the number of positions 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 of a letter by a shift n can be described mathematically as.

### **A.CEASERCIPHER**

### **SOURCE CODE:-**

```
// import required classes and package, if any
import java.util.Scanner;

// create class CaesarCipherExample for encryption and decryption
public class CaesarCipherExample
{
    // ALPHABET string denotes alphabet from a-z
    public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    // create encryptData() method for encrypting user input string with given shift key
    public static String encryptData(String inputStr, int shiftKey)
    {
        // convert inputStr into lower case
        inputStr = inputStr.toLowerCase();

        // encryptStr to store encrypted data
        String encryptStr = "";

        // use for loop for traversing each character of the input string
        for (int i = 0; i < inputStr.length(); i++)
```

```

{
    // get position of each character of inputStr in ALPHABET
    int pos = ALPHABET.indexOf(inputStr.charAt(i));
    // get encrypted char for each char of inputStr
    int encryptPos = (shiftKey + pos) % 26;
    char encryptChar = ALPHABET.charAt(encryptPos);
    // add encrypted char to encrypted string
    encryptStr += encryptChar;
}

// return encrypted string
return encryptStr;
}

// create decryptData() method for decrypting user input string with given shift key
public static String decryptData(String inputStr, int shiftKey)
{
    // convert inputStr into lower case
    inputStr = inputStr.toLowerCase();
    // decryptStr to store decrypted data
    String decryptStr = "";
    // use for loop for traversing each character of the input string
    for (int i = 0; i < inputStr.length(); i++)
    {
        // get position of each character of inputStr in ALPHABET
        int pos = ALPHABET.indexOf(inputStr.charAt(i));
        // get decrypted char for each char of inputStr
        int decryptPos = (pos - shiftKey) % 26;
        // if decryptPos is negative
        if (decryptPos < 0){
            decryptPos = ALPHABET.length() + decryptPos;
        }
        char decryptChar = ALPHABET.charAt(decryptPos);
        // add decrypted char to decrypted string
        decryptStr += decryptChar;
    }
    // return decrypted string

```

```
        return decryptStr;
    }

    // main() method start
    public static void main(String[] args)
    {
        // create an instance of Scanner class
        Scanner sc = new Scanner(System.in);
        // take input from the user
        System.out.println("Enter a string for encryption using Caesar Cipher: ");
        String inputStr = sc.nextLine();
        System.out.println("Enter the value by which each character in the plaintext message gets shifted: ");
        int shiftKey = Integer.valueOf(sc.nextLine());
        System.out.println("Encrypted Data ==> "+encryptData(inputStr, shiftKey));
        System.out.println("Decrypted Data ==> "+decryptData(encryptData(inputStr, shiftKey), shiftKey));
        // close Scanner class object
        sc.close();
    }
}
```

### **OUTPUT:**

**Enter a string for encryption using Caesar Cipher:** hello

**Enter the value by which each character in the plaintext message gets**

**shifted:**

2

Encrypted Data ==> jgnnq

Decrypted Data ==> hello

## **B.SUBSTITUTION CIPHER**

### **DESCRIPTION:**

Hiding some data is known as encryption. When plain text is encrypted it becomes unreadable and is known as cipher text. In a Substitution cipher, any character of plain text from the given fixed set of characters is substituted by some other character from the same set depending on a key. For example with a shift of 1, A would be replaced by B, B would become C, and so on.

### **SOURCE CODE:**

```
// import required classes and package, if any
import java.util.Scanner;

// create class CaesarCipherExample for encryption and decryption
public class CaesarCipherExample
{
    // ALPHABET string denotes alphabet from a-z
    public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    // create encryptData() method for encrypting user input string with given shift key
    public static String encryptData(String inputStr, int shiftKey)
    {
        // convert inputStr into lower case
        inputStr = inputStr.toLowerCase();

        // encryptStr to store encrypted data
        String encryptStr = "";

        // use for loop for traversing each character of the input string
        for (int i = 0; i < inputStr.length(); i++)
        {
            // get position of each character of inputStr in ALPHABET
            int pos = ALPHABET.indexOf(inputStr.charAt(i));

            // get encrypted char for each char of inputStr
            int encryptPos = (shiftKey + pos) % 26;

            char encryptChar = ALPHABET.charAt(encryptPos);

            // add encrypted char to encrypted string
            encryptStr += encryptChar;
        }

        // return encrypted string
        return encryptStr;
    }
}
```

```

}

// create decryptData() method for decrypting user input string with given shift key
public static String decryptData(String inputStr, int shiftKey)
{
    // convert inputStr into lower case
    inputStr = inputStr.toLowerCase();
    // decryptStr to store decrypted data
    String decryptStr = "";
    // use for loop for traversing each character of the input string
    for (int i = 0; i < inputStr.length(); i++)
    {
        // get position of each character of inputStr in ALPHABET
        int pos = ALPHABET.indexOf(inputStr.charAt(i));
        // get decrypted char for each char of inputStr
        int decryptPos = (pos - shiftKey) % 26;
        // if decryptPos is negative
        if (decryptPos < 0){
            decryptPos = ALPHABET.length() + decryptPos;
        }
        char decryptChar = ALPHABET.charAt(decryptPos);
        // add decrypted char to decrypted string
        decryptStr += decryptChar;
    }
    // return decrypted string
    return decryptStr;
}

// main() method start
public static void main(String[] args)
{
    // create an instance of Scanner class
    Scanner sc = new Scanner(System.in);
    // take input from the user
    System.out.println("Enter a string for encryption using Caesar Cipher: ");
    String inputStr = sc.nextLine();

```



```
System.out.println("Enter the value by which each character in the plaintext message gets shifted: ");
int shiftKey = Integer.valueOf(sc.nextLine());
System.out.println("Encrypted Data ==> "+encryptData(inputStr, shiftKey));
System.out.println("Decrypted Data ==> "+decryptData(encryptData(inputStr, shiftKey), shiftKey));
// close Scanner class object
sc.close();
}
}
```

### **OUTPUT:**

**Enter the plain text:**

hello

**Cipher Text is:** lipps

**Recovered plain text:** hello

## **C.HILL CIPHER**

### **DESCRIPTION:**

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  $n \times n$  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).

### **SOURCE CODE:**

```
import java.util.ArrayList;
import java.util.Scanner;
public class HillCipherExample {
    //method to accept key matrix
    private static int[][] getKeyMatrix() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter key matrix:");
        String key = sc.nextLine();
        //int len = key.length();
        double sq = Math.sqrt(key.length());
        if (sq != (long) sq) {
            System.out.println("Cannot Form a square matrix");
        }
        int len = (int) sq;
        int[][] keyMatrix = new int[len][len];
        int k = 0;
        for (int i = 0; i < len; i++)
        {
            for (int j = 0; j < len; j++)
            {
                keyMatrix[i][j] = ((int) key.charAt(k)) - 97;
                k++;
            }
        }
    }
}
```

```

return keyMatrix;
}

// Below method checks whether the key matrix is valid (det=0)
private static void isValidMatrix(int[][] keyMatrix) {
    int det = keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] *
keyMatrix[1][0];
    // If det=0, throw exception and terminate
    if(det == 0) { throw new java.lang.Error("Det equals to zero, invalid key matrix!");
    }
}

// This method checks if the reverse key matrix is valid (matrix mod26 = (1,0,0,1)
private static void isValidReverseMatrix(int[][] keyMatrix,
int[][] reverseMatrix) {
    int[][] product = new int[2][2];
    // Find the product matrix of key matrix times reverse key matrix
    product[0][0] = (keyMatrix[0][0]*reverseMatrix[0][0] +
keyMatrix[0][1] * reverseMatrix[1][0]) % 26;
    product[0][1] = (keyMatrix[0][0]*reverseMatrix[0][1] +
keyMatrix[0][1] * reverseMatrix[1][1]) % 26;
    product[1][0] = (keyMatrix[1][0]*reverseMatrix[0][0] +
keyMatrix[1][1] * reverseMatrix[1][0]) % 26;
    product[1][1] = (keyMatrix[1][0]*reverseMatrix[0][1] +
keyMatrix[1][1] * reverseMatrix[1][1]) % 26;
    // Check if a=1 and b=0 and c=0 and d=1
    // If not, throw exception and terminate
    if(product[0][0] != 1 || product[0][1] != 0 || product[1][0] != 0
|| product[1][1] != 1) {
        throw new java.lang.Error("Invalid reverse matrix found!");
    }
}

// This method calculates the reverse key matrix
private static int[][] reverseMatrix(int[][] keyMatrix) {
    int detmod26 = (keyMatrix[0][0] * keyMatrix[1][1] -
keyMatrix[0][1] * keyMatrix[1][0]) % 26; // Calc det
    int factor;

```

```

int[][] reverseMatrix = new int[2][2];
// Find the factor for which is true that
// factor*det = 1 mod 26
for(factor=1; factor < 26; factor++)
{
    if((detmod26 * factor) % 26 == 1)
    {
        break;
    }
}

// Calculate the reverse key matrix elements using the factor found
reverseMatrix[0][0] = keyMatrix[1][1] * factor % 26;
reverseMatrix[0][1] = (26 - keyMatrix[0][1]) * factor % 26;
reverseMatrix[1][0] = (26 - keyMatrix[1][0]) * factor % 26;
reverseMatrix[1][1] = keyMatrix[0][0] * factor % 26;
return reverseMatrix;
}

// This method echoes the result of encrypt/decrypt
private static void echoResult(String label, int adder,
ArrayList<Integer> phrase) {
    int i;
    System.out.print(label);
    // Loop for each pair
    for(i=0; i < phrase.size(); i += 2) {
        System.out.print(Character.toChars(phrase.get(i) + (64 +
adder)));
        System.out.print(Character.toChars(phrase.get(i+1) + (64 +
adder)));
        if(i+2 < phrase.size()) {
            System.out.print("-");
        }
    }
    System.out.println();
}

// This method makes the actual encryption

```

```

public static void encrypt(String phrase, boolean alphaZero)
{
    int i;
    int adder = alphaZero ? 1 : 0; // For calculations depending on the alphabet
    int[][] keyMatrix;
    ArrayList<Integer> phraseToNum = new ArrayList<>();
    ArrayList<Integer> phraseEncoded = new ArrayList<>();
    // Delete all non-english characters, and convert phrase to upper case
    phrase = phrase.replaceAll("[^a-zA-Z]", "").toUpperCase();
    // If phrase length is not an even number, add "Q" to make it even
    if(phrase.length() % 2 == 1) {
        phrase += "Q";
    }
    // Get the 2x2 key matrix from sc
    keyMatrix = getKeyMatrix();
    // Check if the matrix is valid (det != 0)
    isValidMatrix(keyMatrix);
    // Convert characters to numbers according to their
    // place in ASCII table minus 64 positions (A=65 in ASCII table)
    // If we use A=0 alphabet, subtract one more (adder)
    for(i=0; i < phrase.length(); i++) {
        phraseToNum.add(phrase.charAt(i) - (64 + adder));
    }
    // Find the product per pair of the phrase with the key matrix modulo 26
    // If we use A=1 alphabet and result is 0, replace it with 26 (Z)
    for(i=0; i < phraseToNum.size(); i += 2) {
        int x = (keyMatrix[0][0] * phraseToNum.get(i) +
        keyMatrix[0][1] * phraseToNum.get(i+1)) % 26;
        int y = (keyMatrix[1][0] * phraseToNum.get(i) +
        keyMatrix[1][1] * phraseToNum.get(i+1)) % 26;
        phraseEncoded.add(alphaZero ? x : (x == 0 ? 26 : x ));
        phraseEncoded.add(alphaZero ? y : (y == 0 ? 26 : y ));
    }
    // Print the result
    echoResult("Encoded phrase: ", adder, phraseEncoded);
}

```

```

}
// This method makes the actual decryption
public static void decrypt(String phrase, boolean alphaZero)
{
    int i, adder = alphaZero ? 1 : 0;
    int[][] keyMatrix, revKeyMatrix;
    ArrayList<Integer> phraseToNum = new ArrayList<>();
    ArrayList<Integer> phraseDecoded = new ArrayList<>();
    // Delete all non-english characters, and convert phrase to upper case
    phrase = phrase.replaceAll("[^a-zA-Z]", "").toUpperCase();
    // Get the 2x2 key matrix from sc
    keyMatrix = getKeyMatrix();
    // Check if the matrix is valid (det != 0)
    isValidMatrix(keyMatrix);
    // Convert numbers to characters according to their
    // place in ASCII table minus 64 positions (A=65 in ASCII table)
    // If we use A=0 alphabet, subtract one more (adder)
    for(i=0; i < phrase.length(); i++) {
        phraseToNum.add(phrase.charAt(i) - (64 + adder));
    }
    // Find the reverse key matrix
    revKeyMatrix = reverseMatrix(keyMatrix);
    // Check if the reverse key matrix is valid (product = 1,0,0,1)
    isValidReverseMatrix(keyMatrix, revKeyMatrix);
    // Find the product per pair of the phrase with the reverse key matrix modulo 26
    for(i=0; i < phraseToNum.size(); i += 2) {
        phraseDecoded.add((revKeyMatrix[0][0] * phraseToNum.get(i) +
        revKeyMatrix[0][1] * phraseToNum.get(i+1)) % 26);
        phraseDecoded.add((revKeyMatrix[1][0] * phraseToNum.get(i) +
        revKeyMatrix[1][1] * phraseToNum.get(i+1)) % 26);
    }
    // Print the result
    echoResult("Decoded phrase: ", adder, phraseDecoded);
}

```

```
//main method
public static void main(String[] args) {
    String opt, phrase;
    byte[] p;
    Scanner sc = new Scanner(System.in);
    System.out.println("Hill Cipher Implementation (2x2)");
    System.out.println("-----");
    System.out.println("1. Encrypt text (A=0,B=1,...Z=25)");
    System.out.println("2. Decrypt text (A=0,B=1,...Z=25)");
    System.out.println("3. Encrypt text (A=1,B=2,...Z=26)");
    System.out.println("4. Decrypt text (A=1,B=2,...Z=26)");
    System.out.println();
    System.out.println("Type any other character to exit");
    System.out.println();
    System.out.print("Select your choice: ");
    opt = sc.nextLine();
    switch (opt)
    {
        case "1":
            System.out.print("Enter phrase to encrypt: ");
            phrase = sc.nextLine();
            encrypt(phrase, true);
            break;
        case "2":
            System.out.print("Enter phrase to decrypt: ");
            phrase = sc.nextLine();
            decrypt(phrase, true);
            break;
        case "3":
            System.out.print("Enter phrase to encrypt: ");
            phrase = sc.nextLine();
            encrypt(phrase, false);
            break;
        case "4":
            System.out.print("Enter phrase to decrypt: ");
```

```
phrase = sc.nextLine();
decrypt(phrase, false);
break;
}
}
}
```

### **OUTPUT:**

Hill Cipher Implementation (2x2)

1. Encrypt text (A=0,B=1,...Z=25)
2. Decrypt text (A=0,B=1,...Z=25)
3. Encrypt text (A=1,B=2,...Z=26)
4. Decrypt text (A=1,B=2,...Z=26)

Type any other character to exit

**Select your choice:** 1

**Enter phrase to encrypt:** hello

**Enter key matrix:**

nwkc

**Encoded phrase:** XA-VC-OQ



## **EXPERIMENT: 4**

### **AIM:**

**Write a c/java program to implement the des algorithm logic.**

### **DESCRIPTION:**

Data encryption standard (DES) has been found vulnerable to very powerful attacks and therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

### **SOURCE CODE:**

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

class DESExample {
    Cipher ecipher;
    Cipher dcipher;

    DESExample(SecretKey key) throws Exception {
        ecipher = Cipher.getInstance("DES");
        dcipher = Cipher.getInstance("DES");
        ecipher.init(Cipher.ENCRYPT_MODE, key);
        dcipher.init(Cipher.DECRYPT_MODE, key);
    }

    public String encrypt(String str) throws Exception {
        // Encode the string into bytes using utf-8
        byte[] utf8 = str.getBytes("UTF8");
        // Encrypt
        byte[] enc = ecipher.doFinal(utf8);
        // Encode bytes to base64 to get a string
        return Base64.getEncoder().encodeToString(enc);
    }
}
```

```
public String decrypt(String str) throws Exception {  
    // Decode base64 to get bytes  
    byte[] dec = Base64.getDecoder().decode(str);  
    byte[] utf8 = dcipher.doFinal(dec);  
    // Decode using utf-8  
    return new String(utf8, "UTF8");  
}  
  
public static void main(String[] argv) throws Exception {  
    Scanner myObj=new Scanner(System.in);  
    System.out.println("enter the plain text:");  
    final String secretText = myObj.nextLine();  
    SecretKey key = KeyGenerator.getInstance("DES").generateKey();  
    DESEExample encrypter = new DESEExample(key);  
    String encrypted = encrypter.encrypt(secretText);  
    System.out.println("Encrypted Value: " + encrypted);  
    String decrypted = encrypter.decrypt(encrypted);  
    System.out.println("Decrypted: " + decrypted);  
}  
}
```

### **OUTPUT:**

**Enter the plain text:**

hello

**Encrypted Value:** nOCqkWAAwCY=

Decrypted: hello

## **EXPERIMENT: 5**

### **AIM:**

**Write a c/java program to implement the blowfish algorithm logic.**

### **DESCRIPTION:**

Blowfish is an encryption technique designed by Bruce Schneier in 1993 as an alternative to DES Encryption Technique. It is significantly faster than DES and provides a good encryption rate with no effective cryptanalysis technique found to date. It is one of the first, secure block cyphers not subject to any patents and hence freely available for anyone to use.

- Block Size: 64-bits
- Key Size: 32-bits to 448-bits variable size
- number of sub keys: 18 [P-array]
- number of rounds: 16
- number of substitution boxes: 4 [each having 512 entries of 32-bits each]

### **SOURCE CODE:**

```
import java.util.Scanner;
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

/**
 * This program demonstrates how to encrypt/decrypt
 * input using the Blowfish
 * Cipher with the Java Cryptography.
 */
public class BlowfishDemo {
    public String encrypt(String password, String key) throws
        NoSuchAlgorithmException, NoSuchPaddingException,
        InvalidKeyException, IllegalBlockSizeException,
```

```

BadPaddingException, UnsupportedEncodingException {
byte[] KeyData = key.getBytes();
SecretKeySpec KS = new SecretKeySpec(KeyData, "Blowfish");
Cipher cipher = Cipher.getInstance("Blowfish");
cipher.init(Cipher.ENCRYPT_MODE, KS);
String encryptedtext = Base64.getEncoder().
encodeToString(cipher.doFinal(password.getBytes("UTF-8")));
return encryptedtext;
}

public String decrypt(String encryptedtext, String key)
throws NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {
byte[] KeyData = key.getBytes();
SecretKeySpec KS = new SecretKeySpec(KeyData, "Blowfish");
byte[] encryptedtexttobytes = Base64.getDecoder().
decode(encryptedtext);
Cipher cipher = Cipher.getInstance("Blowfish");
cipher.init(Cipher.DECRYPT_MODE, KS);
byte[] decrypted = cipher.doFinal(encryptedtexttobytes);
String decryptedString =
new String(decrypted, Charset.forName("UTF-8"));
return decryptedString;
}

public static void main(String[] args) throws Exception {
Scanner myObj=new Scanner(System.in);
System.out.println("enter the password:");
final String password = myObj.nextLine();
System.out.println("enter the key:");
final String key = myObj.nextLine();
System.out.println("Password: " + password);
BlowfishDemo obj = new BlowfishDemo();
String enc_output = obj.encrypt(password, key);
System.out.println("Encrypted text: " + enc_output);
String dec_output = obj.decrypt(enc_output, key);

```

```
System.out.println("Decrypted text: " + dec_output);  
}  
}
```

### **OUTPUT:**

```
java -cp /tmp/aSIDN7Fo95 BlowfishDemo
```

**Enter the password:**

hello

**Enter the key:**

hi

**Password:** helloEncrypted text: fgeynH509N0=

Decrypted text: hello

## **EXPERIMENT: 6**

### **AIM:**

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

### **DESCRIPTION:**

The Data Encryption Standard method used for encryption was not promising good security that led to the invention of a highly secure tool called Rijndael Key by Vincent Rijmen and Joan Daemon.

#### **Rijndael Key:**

Rijndael is based on the block cipher method which uses a symmetric key encryption technique. It works with the help of invertible and discrete layers.

- Linear Mix Transform
- Non – Linear Transform
- Key Addition Transform

### **PROGRAM:**

```
import javax.crypto.*;
import javax.crypto.spec.*;

public class Rijndael
{
    private static final String ALGORITHM = "AES";
    private static final int BLOCK_SIZE = 16; // 16 bytes = 128 bits
    private static final int KEY_SIZE = 16; // 16 bytes = 128 bits
    public static byte[] encrypt(byte[] plaintext, byte[] key) throws Exception
    {
        SecretKeySpec secretKey = new SecretKeySpec(key, ALGORITHM);
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encrypted = cipher.doFinal(plaintext);
        return encrypted;
    }
    public static byte[] decrypt(byte[] ciphertext, byte[] key) throws Exception
    {
        SecretKeySpec secretKey = new SecretKeySpec(key, ALGORITHM);
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
```

```
byte[] decrypted = cipher.doFinal(ciphertext);
return decrypted;
}

public static void main(String[] args) throws Exception
{
byte[] plaintext = "Hello, world!".getBytes();
byte[] key = "abcdefghijklmnop".getBytes(); // 16-byte key
byte[] ciphertext = encrypt(plaintext, key);
System.out.println("Ciphertext: " + new String(ciphertext));
byte[] decrypted = decrypt(ciphertext, key);
System.out.println("Decrypted text: " + new String(decrypted));
}
```

### **OUTPUT:**

**Ciphertext:** x????f?x ?/???

**Decrypted text:** Hello, world!

## **EXPERIMENT: 7**

### **AIM:**

**Write the RC4 logic in java cryptography encrypt the text “Hello World” using Blowfish. Create your own key using java key tool.**

### **DESCRIPTION:**

Blowfish.java generates the symmetric key using Blowfish algorithm. Key size assigned here is 128 bits. It works for key size of 256 and 448 bits also. Encryption and decryption method is written based on Blowfish algorithm. Message to encrypt can be given as input. Encrypted and decrypted text is displayed in message dialog.

### **PROGRAM:**

```
import javax.crypto.*;
import javax.crypto.spec.*;

public class RC4
{
    public static byte[] encrypt(byte[] plaintext, byte[] key) throws Exception {
        Cipher cipher = Cipher.getInstance("RC4");
        SecretKeySpec secretKey = new SecretKeySpec(key, "RC4");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encrypted = cipher.doFinal(plaintext);
        return encrypted;
    }

    public static byte[] decrypt(byte[] ciphertext, byte[] key) throws Exception {
        Cipher cipher = Cipher.getInstance("RC4");
        SecretKeySpec secretKey = new SecretKeySpec(key, "RC4");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decrypted = cipher.doFinal(ciphertext);
        return decrypted;
    }

    public static void main(String[] args) throws Exception {
        byte[] plaintext = "Hello, world!".getBytes();
        byte[] key = "abcdefghijklmnop".getBytes(); // 16-byte key
        byte[] ciphertext = encrypt(plaintext, key);
        System.out.println("Ciphertext: " + new String(ciphertext));
    }
}
```



```
byte[] decrypted = decrypt(ciphertext, key);  
System.out.println("Decrypted text: " + new String(decrypted));  
}  
}
```

### **OUTPUT:**

**Cipher text:** /??j?l ?S??

**Decrypted text:** Hello, world!

## **EXPERIMENT: 8**

### **AIM:**

**Write a Java program to implement RSA Algorithm.**

### **DESCRIPTION:**

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

An example of asymmetric cryptography:

1. A client (for example browser) sends its public key to the server and requests some data.
2. The server encrypts the data using the client's public key and sends the encrypted data.
3. The client receives this data and decrypts it.

### **PROGRAM:**

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

public class RSA
{
    private final static BigInteger one = new BigInteger("1");
    private final static SecureRandom random = new SecureRandom();
    private BigInteger privateKey;
    private BigInteger publicKey;
    private BigInteger modulus;
    public RSA(int bitLength) {
        BigInteger p = BigInteger.probablePrime(bitLength / 2, random);
        BigInteger q = BigInteger.probablePrime(bitLength / 2, random);
        BigInteger phi = (p.subtract(one)).multiply(q.subtract(one));
        modulus = p.multiply(q);
        publicKey = new BigInteger("65537");
        privateKey = publicKey.modInverse(phi);
    }
    public BigInteger encrypt(BigInteger message)
    {
        return message.modPow(publicKey, modulus);
    }
}
```

```
public BigInteger decrypt(BigInteger encryptedMessage)
{
    return encryptedMessage.modPow(privateKey, modulus);
}

public static void main(String[] args) {
    RSA rsa = new RSA(1024);
    BigInteger message = new BigInteger("123456789");
    BigInteger encrypted = rsa.encrypt(message);
    System.out.println("Encrypted message: " + encrypted);
    BigInteger decrypted = rsa.decrypt(encrypted);
    System.out.println("Decrypted message: " + decrypted);
}
}
```

### **OUTPUT:**

#### **Encrypted message:**

516146225420234232422515017317092177534123977144793675199204580745325251738166947957  
337514266940976104779928157764064327771748758651702633249896393276012780201647070906  
160356666184000029138502132269476635368978386120245365495700657013727308867388546460  
5576153071520419421559072317863242813325230686772

**Decrypted message:** 123456789

## **EXPERIMENT: 9**

### **AIM:**

**Write a JAVA program to implement the Diffie-Hellman Key Exchange mechanism using HTML and java script. Consider the end user as one of the parties (Alice) and the java script application as other party (bob).**

### **DESCRIPTION:**

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

### **PROGRAM:**

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

public class DiffieHellman {
    private static final BigInteger PRIME = new BigInteger("11223344556677889900"
        + "998877665544332211", 16);
    private static final BigInteger GENERATOR = new BigInteger("2");

    public static void main(String[] args) {
        SecureRandom random = new SecureRandom();
        BigInteger a = new BigInteger(PRIME.bitLength(), random);
        BigInteger A = GENERATOR.modPow(a, PRIME);
        System.out.println("Alice's secret key a: " + a);
        System.out.println("Alice's public key A: " + A);

        // Bob's side
        BigInteger b = new BigInteger(PRIME.bitLength(), random);
        BigInteger B = GENERATOR.modPow(b, PRIME);
        System.out.println("Bob's secret key b: " + b);
        System.out.println("Bob's public key B: " + B);

        // Alice computes shared secret key
```

```
BigInteger s1 = B.modPow(a, PRIME);
System.out.println("Alice's shared secret key: " + s1);
// Bob computes shared secret key
BigInteger s2 = A.modPow(b, PRIME);
System.out.println("Bob's shared secret key: " + s2);
}
}
```

### **OUTPUT:**

**Alice's secret key a:** 699760886692006108126318239687724056660572493

**Alice's public key A:** 17160265641645386768447810365836340556555597

**Bob's secret key b:** 415977760528358312505500195305710448189605869

**Bob's public key B:** 22704366366081328600447601540189720186048117

**Alice's shared secret key:** 162284825122548525444677068678780760593450472

**Bob's shared secret key:** 162284825122548525444677068678780760593450472

### **AIM:**

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

### **DESCRIPTION:**

SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long. It is a U.S. Federal Information Processing Standard and was designed by the United States National Security Agency. SHA-1 is now considered insecure since 2005. Major tech giants browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017. To calculate cryptographic hashing value in Java, MessageDigest Class is used, under the package java.security. MessageDigest Class provides following cryptographic hash function to find hash value of a text as follows:

- MD2
- MD5
- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

### **PROGRAM:**

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

public class SHA1Example {
    public static void main(String[] args) throws NoSuchAlgorithmException {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the text to hash: ");
        String text = scanner.nextLine();
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        md.update(text.getBytes());
        byte[] digest = md.digest();
        StringBuilder sb = new StringBuilder();
        for (byte b : digest)
        {
            sb.append(String.format("%02x", b & 0xff));
        }
    }
}
```

```
}  
System.out.println("SHA-1 hash of " + text + " is: " + sb.toString());  
}  
}
```

### **OUTPUT:**

**Enter the text to hash:** hello world

**SHA-1 hash of hello world is:** 2aae6c35c94fcfb415dbe95f408b9ce91ee846e

## **AIM:**

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

## **DESCRIPTION:**

MD5 is a cryptographic hash function algorithm that takes the message as input of any length and changes it into a fixed-length message of 16 bytes. MD5 algorithm stands for the message-digest algorithm. MD5 was developed as an improvement of MD4, with advanced security purposes. The output of MD5 (Digest size) is always 128 bits. MD5 was developed in 1991 by Ronald Rivest.

## **Use of MD5 Algorithm:**

- It is used for file authentication.
- In a web application, it is used for security purposes. e.g. Secure password of users etc.
- Using this algorithm, We can store our password in 128 bits format.

## **PROGRAM:**

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

public class MD5Example {
    public static void main(String[] args) throws NoSuchAlgorithmException {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the text to hash: ");

        String text = scanner.nextLine();

        MessageDigest md = MessageDigest.getInstance("MD5");

        md.update(text.getBytes());

        byte[] digest = md.digest();

        StringBuilder sb = new StringBuilder();

        for (byte b : digest)
        {
            sb.append(String.format("%02x", b & 0xff));
        }

        System.out.println("MD5 hash of " + text + " is: " + sb.toString());
    }
}
```

## **OUTPUT:**

**Enter the text to hash:** hello world

**MD5 hash of hello world is:** 5eb63bbbe01eed093cb22bb8f5acdc3





