LABORATORY RECORD

NAME: *SAI VISHAL UPPALA*

ROLL NO: *160120733048* **BRANCH & SECTION:** CSE (C1)

ACADEMIC YEAR: 2023-24 **CLASS & SEMESTER:** CSE-1 (VII)

COURSE WITH CODE: CRYPTOGRAPHY AND NETWORK SECURITY LAB(20CSC31)

DEPARTMENT: *COMPUTER SCIENCE AND ENGINEERING*

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE, Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

Certified that this is the bonafide record of the practical work done by the candidate

Mr/Ms. ……SAI VISHAL UPPALA…….. Roll

No:……160120733048..…… of Program .....BE.....

Section…….CSE-1…, Semester …VII……..in the Laboratory course with Code

..Cryptography and Network Security Lab (20CSC31)… during the academic

year…2023-24……

Total Number of Experiments prescribed: ……

Total Number of Experiments done: ……

Signature of the Faculty **HoD**

Semester End Examination held on……

Internal Examiner **External Examiner**

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE, Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in

Vision of Institute

To be the Centre of Excellence in Technical Education and Research.

Mission of Institute

To address the Emerging needs through Quality Technical Education and Advanced Research.

Quality Policy

CBIT imparts value based Technical Education and Training to meet the requirements of students, Industry, Trade/ Profession, Research and Development Organizations for Self-sustained growth of Society.

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE, Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

To be in the frontiers of Computer Science and Engineering with academic excellence and Research.

Mission of the Department

The mission of the Computer Science and Engineering Department is to:

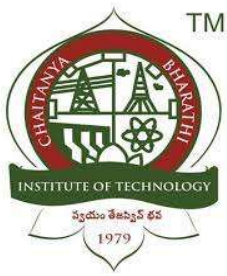
1. *Educate students with the best practices of Computer Science by integrating the latest res into the curriculum*
2. *Develop professionals with sound knowledge in theory and practice of Computer Science Engineering*
3. *Facilitate the development of academia-industry collaboration and societal outreach program*
4. *Prepare students for full and ethical participation in a diverse society and encourage lif learning*

Program Educational Objectives (PEOs)

1. *Practice their profession with confidence by applying new ideas and technologies for the sustainable growth of Industry & Society.*

2. *To pursue higher studies for professional growth with superior ethics & Character.*

3. *Engage in Research leading to innovations/products or become a successful Entrepreneur.*



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE,
Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Outcomes (POs)

PO1. Apply the knowledge of mathematics, science, engineering lamentals, and an engineering specialization for the solution of plex engineering problems

PO2. Identify, formulate, research literature, and analyze comp neering problems reaching substantiated conclusions using first princi athematics, natural sciences, and engineering sciences.

PO3. Design solutions for complex engineering problems and des em components or processes that meet the specified needs with appropr sideration for public health and safety, and cultural, societal, ronmental considerations

PO4. Use research-based knowledge and research methods includ gn of experiments, analysis and interpretation of data, and synthesis of rmation to provide valid conclusions

PO5. Create, select, and apply appropriate techniques, resources, ern engineering and IT tools, including prediction and modelling to comp neering activities, with an understanding of the limitations.

PO6. Apply reasoning informed by the contextual knowledge to ass etal, health, safety, legal, and cultural issues and the consequ onsibilities relevant to the professional engineering practice.

PO7. Understand the impact of the professional engineering solution etal and environmental contexts, and demonstrate the knowledge of, l for sustainable development.

PO8. Apply ethical principles and commit to professional ethics onsibilities and norms of the engineering practice.

PO9. Function effectively as an individual, and as a member or leade rse teams, and in multidisciplinary settings.

PO10. Communicate effectively on complex engineering activities with neering community and with the society at large, such as, being abl

prehend and write effective reports and design documentation, m
tive presentations, and give and receive clear instructions

PO11. Demonstrate knowledge and understanding of the engineering
agement principles and apply these to one's own work, as a member
er in a team, to manage projects and in multidisciplinary environments

PO12. Recognize the need for, and have the preparation and ability
ge in independent and life-long learning in the broadest contex
nological change

Program Specific Outcomes (PSOs)

PSO1. Graduates will acquire the practical competency in Compu
Science and Engineering through emerging technologies and open-sou
platforms related to the domains

PSO2. Graduates will design and develop innovative products by apply
principles of computer science and engineering

PSO3. Graduates will be able to successfully pursue higher educatio
reputed institutions and provide solutions as entrepreneurs.

PSO4. Graduates will be able to work in multidisciplinary teams for ca
growth by exhibiting work ethics and values

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE, Accredited by NAAC with A++ Grade and Programs Accredited by NBA)
Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana
www.cbit.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Name of the Laboratory Course with Code:
Cryptography and Network Security Lab (20CSC31)

Course Outcomes (COs):
CO1. Identify basic security attacks and services

CO2. Design symmetric and asymmetric key algorithms for cryptography

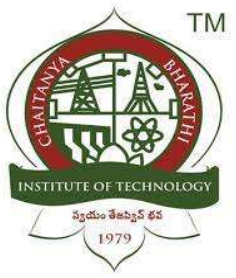
CO3. Create and use of Authentication functions

CO4. Identify and investigate network security threat

CO5. Analyze and design network security protocols

CO-PO/PSO Articulation Matrix:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1															
CO2															
CO3															
CO4															
CO5															



CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Osmania University, Approved by AICTE, Accredited by NAAC with A++ Grade and Programs Accredited by NBA)

Chaitanya Bharathi Post, Gandipet, Kokapet (Vill.), Hyderabad, Ranga Reddy - 500 075, Telangana

www.cbit.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDEX

Exp. No	Name of the Experiment	Date of Experiment	Date of Submission	Page No.	Record Marks/Grade	Signature of Faculty

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: In this program, we are trying to XOR each character in the given string with 0. XOR is a bitwise operator, and it stands for "exclusive or." It performs logical operation. If input bits are the same, then the output will be false(0) else true(1).

ALGORITHM:

Step-1: Start
Step-2: Initialize two character arrays: 'str' for the plain text and 'str1' for the cipher text.
Step-3: Calculate the length of the plain text using the 'strlen' function.
Step-4: Iterate through each character of the plain text using a loop.
    Step-4.1: Print each character of the plain text to display the original message.
Step-5: After printing the original message, iterate through each character of the plain text again.
    Step-5.1: Apply a bitwise XOR operation (^) with 0 to each character and store the result in the corresponding position of the 'str1' array.
    Step-5.2: Print each character of the cipher text to display the encrypted message.
Step-6: End

PROGRAM:

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

```
        printf("\nThe Cipher Text: ");
        for(i=0;i<len;i++)
        {
                str1[i]=str[i]^0;
                printf("%c",str1[i]);
        }
        printf("\n");
}
```

OUTPUT:

```
The Plain Text: Hello World
The Cipher Text: Hello World

--------------------------------
Process exited after 0.04719 seconds with return value 10
Press any key to continue . . .
```

CONCLUSION:

By executing the above, we have successfully performed XOR operation of the given string with 0 and displayed the result.

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 displays the result

DESCRIPTION: In this program, we are trying to AND, OR and XOR each character in the given string with 127.
AND operator is represented as the '&&' double ampersand symbol. It is a logical operator. It checks the condition of two or more operands by combining in an expression, and if all the conditions are true, the logical AND operator returns the Boolean value true or 1. Else it returns false or 0.
The logical OR operator (||) returns the boolean value true if either or both operands is true and returns false otherwise.
XOR is a bitwise operator, and it stands for "exclusive or." It performs logical operation. If input bits are the same, then the output will be false(0) else true(1).

ALGORITHM:
Step-1: Start
Step-2: Initialize a character array 'str' with the text "Hello World" and three additional character arrays 'str1', 'str2', and 'str3' to store the results of the bitwise operations.
Step-3: Calculate the length of the input string using the 'strlen' function.
Step-4: Iterate through each character of the input string and perform the following operations:
Step-4.1: AND Operation:
Step-4.1.1: Apply a bitwise AND operation (&) with the character and 127 (binary: 01111111) and store the result in the corresponding position of the 'str1' array.
Step-4.1.2: Print each character of the result to display the cipher text after the AND operation.
Step-4.2: XOR Operation:
Step-4.2.1: Apply a bitwise XOR operation (^) with the character and 127 and store the result in the corresponding position of the 'str3' array.
Step-4.2.2: Print each character of the result to display the cipher text after the XOR operation.
Step-4.3: OR Operation:

Step-4.3.1: Apply a bitwise OR operation (|) with the character and 127 and store the result in the corresponding position of the 'str2' array.

Step-4.3.2: Print each character of the result to display the cipher text after the OR operation.

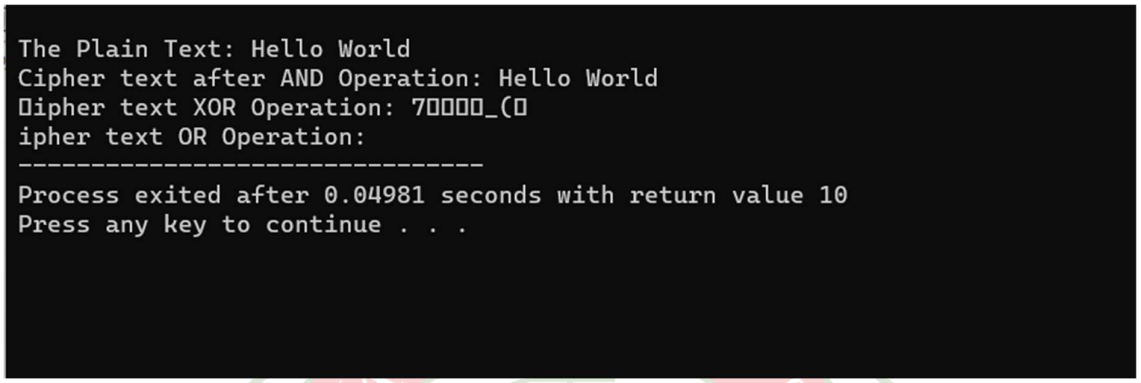
Step-5: End

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
	char str[]="Hello World";
	char str1[11];
	char str2[11];
	char str3[11];
	int i,len;
	len = strlen(str);
	printf("The Plain Text: ");
	for (i=0;i<len;i++)
	{
		printf("%c", str[i]);
	}
	printf("\nCipher text after AND Operation: ");
	for(i=0;i<len;i++)
	{
		str1[i] = str[i]&127;
		printf("%c",str1[i]);
	}
	printf("\nCipher text XOR Operation: ");
	for(i=0;i<len;i++)
	{
		str3[i] = str[i]^127;
		printf("%c",str3[i]);
	}
	printf("\nCipher text OR Operation: ");
	for(i=0;i<len;i++)
	{
		str2[i] = str[i]|127;
		printf("%c",str2[i]);
	}
```

```
        printf("\n");
}
```

OUTPUT:



CONCLUSION:

By executing the above, we have successfully performed AND, OR and XOR operation of the given string with 127 and displayed the result.

AIM: Write a C program to perform encryption and decryption using Ceaser Cipher

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using Ceaser Cipher algorithm.
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.

ALGORITHM:
Step-1: Start
Step-2: Initialize a character array 'text' to store the input message and variables 'ch' and 'key'.
Step-3: Prompt the user to enter a message to encrypt and the key.
Step-4: Encrypt the message:
Step-4.1: Iterate through each character of the input message.
Step-4.2: Check if the character is alphanumeric using isalnum function.
Step-4.3: If alphanumeric, apply the following transformations:
Step-4.3.1: For lowercase letters, shift the character by the key value using modulo arithmetic to ensure looping within the alphabet range.
Step-4.3.2: For uppercase letters, perform the same shift operation.
Step-4.3.3: For digits, shift the character by the key value within the digit range (0-9).
Step-4.4: Update the text array with the encrypted character.
Step-4.5: If the character is not alphanumeric, print "Invalid Message."
Step-5: Print the encrypted message.
Step-6: Decrypt the message:
Step-6.1: Iterate through each character of the encrypted message.
Step-6.2: Check if the character is alphanumeric using isalnum function.

Step-6.3: If alphanumeric, perform the reverse shift operation to decrypt the character.

Step-6.4: Update the text array with the decrypted character.

Step-6.5: If the character is not alphanumeric, print "Invalid Message."

Step-7: Print the decrypted message.

Step-8: End

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
int main()
{
    char text[500], ch;
    int i;
    int key;
    printf("Enter a message to encrypt: ");
    scanf("%s", text);
    printf("Enter the key: ");
    scanf("%d", & key);
    for (i = 0; text[i] != '\0'; ++i)
        {
        ch = text[i];
        if (isalnum(ch)) {
            if (islower(ch)) {
                ch = (ch - 'a' + key) % 26 + 'a';
            }
            if (isupper(ch)) {
                ch = (ch - 'A' + key) % 26 + 'A';
            }
            if (isdigit(ch)) {
                ch = (ch - '0' + key) % 10 + '0';
            }
        }
        else {
            printf("Invalid Message");
        }
        text[i] = ch;
    }
    printf("Encrypted message: %s\n", text);
    for (i = 0; text[i] != '\0'; ++i) {
        ch = text[i];
```

```
        if (isalnum(ch)) {
            if (islower(ch)) {
                ch = (ch - 'a' - key) % 26 + 'a';
            }
            if (isupper(ch)) {
                ch = (ch - 'A' - key) % 26 + 'A';
            }
            if (isdigit(ch)) {
                ch = (ch - '0' - key) % 10 + '0';
            }
        }
        else {
            printf("Invalid Message");
        }
        text[i] = ch;
    }
    printf("Decrypted message: %s", text);
    return 0;
}
```

OUTPUT:

```
Enter a message to encrypt: helloworld
Enter the key: 3
Encrypted message: khoorzruog
Decrypted message: helloworld
--------------------------------
Process exited after 9.582 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using Ceaser Cipher.

AIM: Write a C program to perform encryption and decryption using Substitution Cipher

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using Substitution Cipher algorithm.
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.

ALGORITHM:
Step-1: Start
Step-2: Define functions encrypt and decrypt for encryption and decryption processes.
Step-3: In the encrypt function:
Step-3.1: Iterate through each character of the input message.
Step-3.2: If the character is a lowercase letter, replace it with the corresponding character from the provided key.
Step-4: In the decrypt function:
Step-4.1: Iterate through each character of the input message.
Step-4.2: For each character, iterate through the key to find the matching character and replace it with the corresponding lowercase letter.
Step-5: In the main function:
Step-5.1: Prompt the user to enter a substitution key consisting of 26 lowercase letters.
Step-5.2: Check the length and validity of the key.
Step-5.3: Prompt the user to enter the message to encrypt.
Step-5.4: Call the encrypt function to encrypt the message using the provided key.
Step-5.5: Print the encrypted message.
Step-5.6: Call the decrypt function to decrypt the message using the same key.
Step-5.7: Print the decrypted message.
Step-6: End

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void encrypt(char *message, char *key)
{
    int i;
        for (i = 0; i < strlen(message); i++)
        {
        if (message[i] >= 'a' && message[i] <= 'z')
                {
            message[i] = key[message[i] - 'a'];
        }
    }
}
void decrypt(char *message, char *key)
{
        int i,j;
    for (i = 0; i < strlen(message); i++)
        {
        for (j = 0; j < 26; j++)
                {
            if (message[i] == key[j])
                    {
                message[i] = 'a' + j;
                break;
            }
        }
    }
}
int main()
{
    char key[26];
    int i;
    printf("Enter the substitution key (26 lowercase letters in
random order): ");
    scanf("%s", key);
    if (strlen(key) != 26)
        {
        printf("Invalid key length. Please provide 26 letters.\n");
        return 1;
    }
```

```
    for (i = 0; i < 26; i++)
        {
        if (key[i] < 'a' || key[i] > 'z')
            {
            printf("Invalid key. Please provide only lowercase
letters.\n");
            return 1;
        }
    }
    char message[100];
    printf("Enter the message to encrypt: ");
    scanf(" %[^\n]s", message);
    encrypt(message, key);
    printf("Encrypted message: %s\n", message);
    decrypt(message, key);
    printf("Decrypted message: %s\n", message);
}
```

OUTPUT:

```
Enter the substitution key (26 lowercase letters in random order): qwertyuiopasdfghjklzxcvbnm
Enter the message to encrypt: helloworld
Encrypted message: itssgvgksr
Decrypted message: helloworld

--------------------------------
Process exited after 16.45 seconds with return value 30
Press any key to continue . . .
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using Substitution Cipher.

AIM: Write a C program to perform encryption and decryption using Play Fair Cipher

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using Play Fair Cipher algorithm.
Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a digraph (a pair of two letters) instead of a single letter.

ALGORITHM:
Step-1: Start
Step-2: Input the key and plaintext.
Step-3: Remove spaces from the key and plaintext.
Step-4: Prepare the plaintext by adding 'x' or 'z' as needed.
Step-5: Generate the key matrix using the provided key.
Step-6: Encrypt the plaintext using the key matrix.
Step-7: Convert the ciphertext to uppercase and print it.
Step-8: Decrypt the ciphertext using the key matrix.
Step-9: Convert the decrypted plaintext to lowercase and print it.
Step-10: End

PROGRAM:
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void Playfair(char str[], char keystr[]) {
    char keyMat[5][5];
    int i;
    char ks = strlen(keystr);
    char ps = strlen(str);
    void toUpperCase(char encrypt[], int ps) {
        for (i= 0; i < ps; i++) {
            if (encrypt[i] > 96 && encrypt[i] < 123)
                encrypt[i] -= 32;
        }
    }
        void toLowerCase(char plain[], int ps)
        {
                int i;
```

```
            for (i = 0; i < ps; i++) {
                if (plain[i] > 64 && plain[i] < 91)
                    plain[i] += 32;
            }
    }
int removeSpaces(char* plain, int ps) {
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}
void createMatrix(char keystr[], int ks, char keyMat[5][5]) {
    int flag = 0, *dict;
    int i,j, k;
    dict = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (keystr[i] != 'j')
            dict[keystr[i] - 97] = 2;
    }
    dict['j' - 97] = 1;
    i = 0;
            j = 0;
    for (k = 0; k < ks; k++) {
        if (dict[keystr[k] - 97] == 2) {
            dict[keystr[k] - 97] -= 1;
            keyMat[i][j] = keystr[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
    for (k = 0; k < 26; k++) {
        if (dict[k] == 0) {
            keyMat[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
```

```
            }
        }
    }
}
void search(char keyMat[5][5], char a, char b, int arr[]) {
    int i, j;
        if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';
    for(i = 0; i < 5; i++) {
        for(j = 0; j < 5; j++) {
            if (keyMat[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyMat[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}
int prep(char str[], int p) {
    int sub = p;
    int i, j;
    for (i = 0; i < sub; i += 2) {
        if(str[i]==str[i+1]){
            for(j=sub; j>i+1; j--){
               str[j]=str[j-1];
            }
            str[i+1]='x';
            sub+=1;
        }
    }
    str[sub]='\0';
    if (sub % 2 != 0) {
        str[sub++] = 'z';
        str[sub] = '\0';
    }
    return sub;
}
```

```
void encrypt(char str[], char keyMat[5][5], int pos) {
    int a[4];
    int i;
    for(i=0; i<pos; i+=2){
        search(keyMat, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyMat[a[0]][(a[1] + 1)%5];
            str[i + 1] = keyMat[a[0]][(a[3] + 1)%5];
        }
        else if (a[1] == a[3]) {
            str[i] = keyMat[(a[0] + 1)%5][a[1]];
            str[i + 1] = keyMat[(a[2] + 1)%5][a[1]];
        }
        else {
            str[i] = keyMat[a[0]][a[3]];
            str[i + 1] = keyMat[a[2]][a[1]];
        }
    }
}
void decrypt(char str[], char keyT[5][5], int ps)
    {
        int i, a[4];
        for (i = 0; i < ps; i += 2) {
            search(keyT, str[i], str[i + 1], a);
            if (a[0] == a[2]) {
                if(a[1]==0){
                    str[i] = keyT[a[0]][4];
                    str[i + 1] = keyT[a[0]][(a[3]-1)%5];
                }
                else if(a[3]==0){
                    str[i] = keyT[a[0]][(a[1] - 1)%5];
                    str[i + 1] = keyT[a[0]][4];
                }
                else{
                    str[i] = keyT[a[0]][(a[1] - 1)%5];
                    str[i + 1] = keyT[a[0]][(a[3]-1)%5];
                }

            }
            else if (a[1] == a[3]) {
                if(a[0]==0){
                    str[i] = keyT[4][a[1]];
```

```
                    str[i + 1] = keyT[(a[2]-1)%5][a[1]];
                }
                else if(a[2]==0){
                    str[i] = keyT[(a[0] - 1)%5][a[1]];
                    str[i + 1] = keyT[4][a[1]];
                }
                else{
                    str[i] = keyT[(a[0] - 1)%5][a[1]];
                    str[i + 1] = keyT[(a[2]-1)%5][a[1]];
                }
            }
            else {
                    str[i] = keyT[a[0]][a[3]];
                    str[i + 1] = keyT[a[2]][a[1]];
            }
        }
    }
    ks = removeSpaces(keystr, ks);
    ps = removeSpaces(str, ps);
    ps = prep(str, ps);
    createMatrix(keystr, ks, keyMat);
    encrypt(str, keyMat, ps);
    toUpperCase(str, ps);
    printf("Cipher text: %s\n", str);
    ks = removeSpaces(keystr, ks);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);
    decrypt(str, keyMat, ps);
    printf("Plain text: %s\n", str);
}
int main() {
    char string[200], keyString[200];
    printf("Enter key: ");
    scanf("%[^\n]s", &keyString);
    printf("Enter plaintext: ");
    scanf("\n");
    scanf("%[^\n]s", &string);
    Playfair(string, keyString);
    return 0;
}
```

OUTPUT:

```
Enter key: gavityfalls
Enter plaintext: attackondawn
Cipher text: VGGVDCPONFUO
Plain text: attackondawn

--------------------------------
Process exited after 10.47 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using Play Fair Cipher.

AIM: Write a C program to perform encryption and decryption using Hill Cipher

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using Hill Cipher algorithm.
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 × 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 × n matrices (modulo 26).

ALGORITHM:
Step-1: Start
Step-2: gcdExtended function: Calculates the greatest common divisor (GCD) using the extended Euclidean algorithm, used to calculate modular inverses.
Step-3: getKeyMatrix function: Retrieves the key matrix (2x2) from the user.
Step-4: isValidMatrix function: Checks if the key matrix is valid for the Hill cipher (non-singular and invertible modulo 26).
Step-5: reverseMatrix function: Calculates the inverse of the key matrix modulo 26.
Step-6: echoResult function: Prints the encrypted or decrypted message in groups of two letters.
Step-7: encrypt function: Takes a message and encrypts it using the Hill cipher. Converts characters to numbers (A=0, B=1, …, Z=25), performs matrix multiplication, and prints the encrypted message.
Step-8: decrypt function: Takes an encrypted message and decrypts it using the Hill cipher. Performs matrix multiplication with the inverse of the key matrix, converts numbers back to characters, and prints the decrypted message.
Step-9: main function:
Step-9.1: Presents a menu for user interaction: encrypt, decrypt, or exit.

Step-9.2: For encryption and decryption, it prompts the user for a message and performs the respective operation.
Step-10: End

PROGRAM:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
int gcdExtended(int a, int b, int *x, int *y) {
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int gcd = gcdExtended(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return gcd;
}
void getKeyMatrix(int keyMatrix[2][2]) {
        int i, j;
    printf("Enter key matrix (4 integers): ");
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            scanf("%d", &keyMatrix[i][j]);
        }
    }
}
void isValidMatrix(int keyMatrix[2][2]) {
    int det = keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] *
keyMatrix[1][0];
    if (det == 0) {
        printf("Det equals to zero, invalid key matrix!\n");
        exit(1);
    }
    int x, y;
    if (gcdExtended(det, 26, &x, &y) != 1) {
        printf("Determinant is not invertible modulo 26, invalid key
matrix!\n");
        exit(1);
```

```
    }
}
void reverseMatrix(int keyMatrix[2][2], int reverseMatrix[2][2]) {
    int det = keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] *
keyMatrix[1][0];
    int detMod26 = (det % 26 + 26) % 26;
    int factor;
    for (factor = 1; factor < 26; factor++) {
        if ((detMod26 * factor) % 26 == 1) {
            break;
        }
    }
    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;
}
void echoResult(const char *label, int adder, int *phrase, int size)
{
        int i;
    printf("%s", label);
    for (i = 0; i < size; i += 4) {
        printf("%c%c", phrase[i] + (64 + adder), phrase[i + 1] + (64 +
adder));
        if (i + 4 < size) {
            printf("-");
        }
    }
    printf("\n");
}
void encrypt(const char *phrase, int alphaZero) {
    int keyMatrix[2][2];
    int i;
    getKeyMatrix(keyMatrix);
    isValidMatrix(keyMatrix);
    int len = strlen(phrase);
    int phraseToNum[len];
    for (i = 0; i < len; i++) {
        phraseToNum[i] = toupper(phrase[i]) - (64 + alphaZero);
    }
    int phraseEncoded[len * 2];
    for (i = 0; i < len; i += 2) {
```

```
        int x = (keyMatrix[0][0] * phraseToNum[i] + keyMatrix[0][1] *
phraseToNum[i + 1]) % 26;
        int y = (keyMatrix[1][0] * phraseToNum[i] + keyMatrix[1][1] *
phraseToNum[i + 1]) % 26;
        phraseEncoded[i * 2] = alphaZero ? x : (x == 0 ? 26 : x);
        phraseEncoded[i * 2 + 1] = alphaZero ? y : (y == 0 ? 26 : y);
    }
    echoResult("Encrypted message: ", alphaZero, phraseEncoded,
len * 2);
}
void decrypt(const char *phrase, int alphaZero) {
    int keyMatrix[2][2], revKeyMatrix[2][2];
    getKeyMatrix(keyMatrix);
    isValidMatrix(keyMatrix);
    int len = strlen(phrase);
    int phraseToNum[len];
    int i;
    for (i = 0; i < len; i++) {
        phraseToNum[i] = toupper(phrase[i]) - (64 + alphaZero);
    }
    reverseMatrix(keyMatrix, revKeyMatrix);
    int phraseDecoded[len * 2];
    for (i = 0; i < len; i += 2) {
        phraseDecoded[i * 2] = (revKeyMatrix[0][0] * phraseToNum[i]
+ revKeyMatrix[0][1] * phraseToNum[i + 1]) % 26;
        phraseDecoded[i * 2 + 1] = (revKeyMatrix[1][0] *
phraseToNum[i] + revKeyMatrix[1][1] * phraseToNum[i + 1]) % 26;
    }
    echoResult("Decrypted message: ", alphaZero, phraseDecoded,
len * 2);
}
int main() {
    char opt;
    char phrase[100];
    printf("Hill Cipher Implementation (2x2)\n");
    printf("------------------------\n");
    printf("1. Encrypt text (A=0,B=1,...Z=25)\n");
    printf("2. Decrypt text (A=0,B=1,...Z=25)\n");
    printf("3. Exit\n\n");
    while(1) {
        printf("Select your choice: ");
        scanf(" %c", &opt);
```

```
        switch (opt) {
        case '1':
              printf("Enter message to encrypt: ");
           scanf("%s", phrase);
                 encrypt(phrase, 1);
                 break;
              case '2':
                 printf("Enter message to decrypt: ");
                 scanf("%s", phrase);
                 decrypt(phrase, 1);
                 break;
              case '3':
              exit(1);
        }
        }
    return 0;
}
```

OUTPUT:

```
Hill Cipher Implementation (2x2)
------------------------
1. Encrypt text (A=0,B=1,...Z=25)
2. Decrypt text (A=0,B=1,...Z=25)
3. Exit

Select your choice: 1
Enter message to encrypt: attack
Enter key matrix (4 integers): 2 3 3 6
Encrypted message: FK-MF-IO
Select your choice: 2
Enter message to decrypt: fkmfio
Enter key matrix (4 integers): 2 3 3 6
Decrypted message: AT-TA-CK
Select your choice: 3

--------------------------------
Process exited after 24.59 seconds with return value 1
Press any key to continue . . .
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using Hill Cipher.

CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
స్వయం తేజస్వినా భవ
1979

AIM: Write a program to implement the DES algorithm logic.

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using DES algorithm.
The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).

ALGORITHM:
Step-1: Start
Step-2: Input: Plain text (in hexadecimal), Encryption key (in hexadecimal)
Step-3: Key Generation:
Step-3.1: Convert the encryption key from hexadecimal to binary.
Step-3.2: Perform an initial permutation on the key using a predefined permutation table (keyp).
Step-3.3: Split the key into two halves, left and right, each containing 28 bits.
Step-3.4: For each of the 16 rounds:
Step-3.4.1: Perform circular left shifts on both the left and right key halves based on a predefined shift table.
Step-3.4.2: Combine the left and right halves.
Step-3.4.3: Perform a key compression permutation using a predefined permutation table (key_comp) to generate the round key.
Step-3.4.4: Store the round key.
Step-4: Initial Permutation (IP):
Step-4.1: Convert the plain text from hexadecimal to binary.
Step-4.3: Perform an initial permutation on the plain text using a predefined permutation table (initial_perm).

Step-5: 16 Rounds of Encryption:

Step-5.1: For each round (1 to 16):

Step-5.1.1: Expand the right half of the plain text using a predefined permutation table (exp_d).

Step-5.1.2: Perform an XOR operation between the expanded right half and the current round key.

Step-5.1.3: Apply S-box substitution using predefined S-boxes and permutation (per).

Step-5.1.4: Perform another permutation (permutation) on the S-box output.

Step-5.1.5: Perform an XOR operation between the left half of the plain text and the result of the previous steps.

Step-5.1.6: Swap the left and right halves, except in the final round.

Step-6: Final Permutation (FP):

Step-6.1: Combine the left and right halves after the 16 rounds.

Step-6.2: Perform a final permutation on the combined result using a predefined permutation table (final_perm).

Step-7: Output: The final combined result after the final permutation is the ciphertext (in binary).

Step-8: Convert the ciphertext from binary to hexadecimal for the final encrypted message.

Step-9: Return the encrypted message (ciphertext) in hexadecimal format.

Step-10: End

PROGRAM:

```
def hex2bin(s):
    mp = {'0': "0000", '1': "0001", '2': "0010", '3': "0011", '4': "0100", '5': "0101", '6': "0110", '7': "0111", '8': "1000", '9': "1001", 'A': "1010", 'B': "1011", 'C': "1100", 'D': "1101", 'E': "1110", 'F': "1111"}
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin
def bin2hex(s):
    mp = {"0000": '0', "0001": '1', "0010": '2', "0011": '3', "0100": '4', "0101": '5', "0110": '6', "0111": '7', "1000": '8', "1001": '9',
```

```
"1010": 'A', "1011": 'B', "1100": 'C', "1101": 'D', "1110": 'E', "1111":
'F'}
        hex = ""
        for i in range(0, len(s), 4):
                ch = ""
                ch = ch + s[i]
                ch = ch + s[i + 1]
                ch = ch + s[i + 2]
                ch = ch + s[i + 3]
                hex = hex + mp[ch]
        return hex
def bin2dec(binary):
        binary1 = binary
        decimal, i, n = 0, 0, 0
        while(binary != 0):
                dec = binary % 10
                decimal = decimal + dec * pow(2, i)
                binary = binary//10
                i += 1
        return decimal
def dec2bin(num):
        res = bin(num).replace("0b", "")
        if(len(res) % 4 != 0):
                div = len(res) / 4
                div = int(div)
                counter = (4 * (div + 1)) - len(res)
                for i in range(0, counter):
                        res = '0' + res
        return res
def permute(k, arr, n):
        permutation = ""
        for i in range(0, n):
                permutation = permutation + k[arr[i] - 1]
        return permutation
def shift_left(k, nth_shifts):
        s = ""
        for i in range(nth_shifts):
                for j in range(1, len(k)):
                        s = s + k[j]
                s = s + k[0]
```

```
            k = s
            s = ""
        return k
def xor(a, b):
        ans = ""
        for i in range(len(a)):
            if a[i] == b[i]:
                ans = ans + "0"
            else:
                ans = ans + "1"
        return ans
initial_perm = [58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28,
20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24,
16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7]
exp_d = [32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12, 13, 12,
13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1]
per = [16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26, 5, 18, 31, 10,
2, 8, 24, 14, 32, 27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25]
sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7], [0, 15,
7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8], [4, 1, 14, 8, 13, 6, 2,
11, 15, 12, 9, 7, 3, 10, 5, 0], [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14,
10, 0, 6, 13]], [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
[3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5], [0, 14, 7, 11,
10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15], [13, 8, 10, 1, 3, 15, 4, 2, 11,
6, 7, 12, 0, 5, 14, 9]], [[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11,
4, 2, 8], [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1], [13, 6,
4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7], [1, 10, 13, 0, 6, 9, 8,
7, 4, 15, 14, 3, 11, 5, 2, 12]], [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5,
11, 12, 4, 15], [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
[10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4], [3, 15, 0, 6, 10,
1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]], [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5,
3, 15, 13, 0, 14, 9], [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9,
8, 6], [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14], [11, 8,
12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]], [[12, 1, 10, 15, 9, 2,
6, 8, 0, 13, 3, 4, 14, 7, 5, 11], [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13,
14, 0, 11, 3, 8], [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
[4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]], [[4, 11, 2, 14,
15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1], [13, 0, 11, 7, 4, 9, 1, 10, 14,
```

```
3, 5, 12, 2, 15, 8, 6], [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5,
9, 2], [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]], [[13, 2,
8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7], [1, 15, 13, 8, 10, 3, 7,
4, 12, 5, 6, 11, 0, 14, 9, 2], [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13,
15, 3, 5, 8], [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]]
final_perm = [40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23,
63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61,
29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,
34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25]
def encrypt(pt, rkb, rk):
        pt = hex2bin(pt)
        pt = permute(pt, initial_perm, 64)
        print("After initial permutation", bin2hex(pt))
        left = pt[0:32]
        right = pt[32:64]
        for i in range(0, 16):
                right_expanded = permute(right, exp_d, 48)
                xor_x = xor(right_expanded, rkb[i])
                sbox_str = ""
                for j in range(0, 8):
                        row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
                        col = bin2dec(
                                int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j
* 6 + 3] + xor_x[j * 6 + 4]))
                        val = sbox[j][row][col]
                        sbox_str = sbox_str + dec2bin(val)
                sbox_str = permute(sbox_str, per, 32)
                result = xor(left, sbox_str)
                left = result
                if(i != 15):
                        left, right = right, left
                print("Round ", i + 1, " ", bin2hex(left),
                        " ", bin2hex(right), " ", rk[i])
        combine = left + right
        cipher_text = permute(combine, final_perm, 64)
        return cipher_text
pt = "123456ABCD132536"
key = "AABB09182736CCDD"
key = hex2bin(key)
```

```
keyp = [57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2,
59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36, 63, 55, 47, 39, 31,
23, 15, 7, 62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21,
13, 5, 28, 20, 12, 4]
key = permute(key, keyp, 56)
shift_table = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
key_comp = [14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12,
4, 26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40, 51,
45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32]
left = key[0:28]
right = key[28:56]
rkb = []
rk = []
for i in range(0, 16):
        left = shift_left(left, shift_table[i])
        right = shift_left(right, shift_table[i])
        combine_str = left + right
        round_key = permute(combine_str, key_comp, 48)
        rkb.append(round_key)
        rk.append(bin2hex(round_key))
print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ", cipher_text)
print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ", text)
```

OUTPUT:

```
Plain Text:  123456ABCD132536
Key:  AABB09182736CCDD
Encryption
After initial permutation 14A7D67818CA18AD
Round  1   18CA18AD   5A78E394   194CD072DE8C
Round  2   5A78E394   4A1210F6   4568581ABCCE
Round  3   4A1210F6   B8089591   06EDA4ACF5B5
Round  4   B8089591   236779C2   DA2D032B6EE3
Round  5   236779C2   A15A4B87   69A629FEC913
Round  6   A15A4B87   2E8F9C65   C1948E87475E
Round  7   2E8F9C65   A9FC20A3   708AD2DDB3C0
Round  8   A9FC20A3   308BEE97   34F822F0C66D
Round  9   308BEE97   10AF9D37   84BB4473DCCC
Round  10   10AF9D37   6CA6CB20   02765708B5BF
Round  11   6CA6CB20   FF3C485F   6D5560AF7CA5
Round  12   FF3C485F   22A5963B   C2C1E96A4BF3
Round  13   22A5963B   387CCDAA   99C31397C91F
Round  14   387CCDAA   BD2DD2AB   251B8BC717D0
Round  15   BD2DD2AB   CF26B472   3330C5D9A36D
Round  16   19BA9212   CF26B472   181C5D75C66D
Cipher Text :  C0B7A8D05F3A829C
Decryption
After initial permutation 19BA9212CF26B472
Round  1   CF26B472   BD2DD2AB   181C5D75C66D
Round  2   BD2DD2AB   387CCDAA   3330C5D9A36D
Round  3   387CCDAA   22A5963B   251B8BC717D0
Round  4   22A5963B   FF3C485F   99C31397C91F
Round  5   FF3C485F   6CA6CB20   C2C1E96A4BF3
Round  6   6CA6CB20   10AF9D37   6D5560AF7CA5
Round  7   10AF9D37   308BEE97   02765708B5BF
Round  8   308BEE97   A9FC20A3   84BB4473DCCC
Round  9   A9FC20A3   2E8F9C65   34F822F0C66D
Round  10   2E8F9C65   A15A4B87   708AD2DDB3C0
Round  11   A15A4B87   236779C2   C1948E87475E
Round  12   236779C2   B8089591   69A629FEC913
Round  13   B8089591   4A1210F6   DA2D032B6EE3
Round  14   4A1210F6   5A78E394   06EDA4ACF5B5
Round  15   5A78E394   18CA18AD   4568581ABCCE
Round  16   14A7D678   18CA18AD   194CD072DE8C
Plain Text :  123456ABCD132536
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using DES algorithm.

AIM: Write a program to implement the Blowfish algorithm logic.

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using Blowfish algorithm.

ALGORITHM:

Step-1: Start

Step-2: Key Initialization:

Step-2.1: Blowfish uses a variable-length key, typically between 32 and 448 bits (4 to 56 bytes).

Step-2.2: The key is expanded into an initial P-array (a set of fixed values) and S-boxes (substitution boxes).

Step-2.3: The initial P-array and S-boxes are used to create an initial state.

Step-3: Data Splitting:

Step-3.1: The data to be encrypted (plaintext) is divided into fixed-size blocks (usually 64 bits or 8 bytes).

Step-3.2: If the last block is not the correct size, it's padded to match the block size.

Step-4: Data Encryption (Block Processing):

Step-4.1: The plaintext block is divided into two halves, left and right.

Step-4.2: A series of iterations (rounds) are performed on the data.

Step-4.3: During each round:

Step-4.3.1: The right half is XORed with a subkey derived from the P-array.

Step-4.3.2: The result is passed through the Blowfish encryption function.

Step-4.3.3: The output of the encryption function is XORed with the left half.

Step-4.3.4: Finally, the left and right halves are swapped.

Step-5: Data Decryption (Block Processing):

Step-5.1: Decryption is essentially the reverse of encryption.

Step-5.2: The ciphertext block is divided into left and right halves.

Step-5.3: A series of iterations are performed, but with subkeys derived from the P-array in reverse order.
Step-6: Finalization:
Step-6.1: After all blocks have been processed, the ciphertext blocks are concatenated to form the encrypted message.
Step-6.2: During decryption, the ciphertext blocks are processed in reverse order to obtain the original plaintext.
Step-10: End

PROGRAM:

```
p = [0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344,
0xA4093822, 0x299F31D0, 0x082EFA98, 0xEC4E6C89,
0x452821E6, 0x38D01377, 0xBE5466CF, 0x34E90C6C,
0xC0AC29B7, 0xC97C50DD, 0x3F84D5B5, 0xB5470917,
0x9216D5D9, 0x8979FB1B]
s = [[0xD1310BA6, 0x98DFB5AC, 0x2FFD72DB, 0xD01ADFB7,
0xB8E1AFED, 0x6A267E96, 0xBA7C9045, 0xF12C7F99,
0x24A19947, 0xB3916CF7, 0x0801F2E2, 0x858EFC16,
0x636920D8, 0x71574E69, 0xA458FEA3, 0xF4933D7E,
0x0D95748F, 0x728EB658, 0x718BCD58, 0x82154AEE,
0x7B54A41D, 0xC25A59B5, 0x9C30D539, 0x2AF26013,
0xC5D1B023, 0x286085F0, 0xCA417918, 0xB8DB38EF,
0x8E79DCB0, 0x603A180E, 0x6C9E0E8B, 0xB01E8A3E,
0xD71577C1, 0xBD314B27, 0x78AF2FDA, 0x55605C60,
0xE65525F3, 0xAA55AB94, 0x57489862, 0x63E81440,
0x55CA396A, 0x2AAB10B6, 0xB4CC5C34, 0x1141E8CE,
0xA15486AF, 0x7C72E993, 0xB3EE1411, 0x636FBC2A,
0x2BA9C55D, 0x741831F6, 0xCE5C3E16, 0x9B87931E,
0xAFD6BA33, 0x6C24CF5C, 0x7A325381, 0x28958677,
0x3B8F4898, 0x6B4BB9AF, 0xC4BFE81B, 0x66282193,
0x61D809CC, 0xFB21A991, 0x487CAC60, 0x5DEC8032,
0xEF845D5D, 0xE98575B1, 0xDC262302, 0xEB651B88,
0x23893E81, 0xD396ACC5, 0x0F6D6FF3, 0x83F44239,
0x2E0B4482, 0xA4842004, 0x69C8F04A, 0x9E1F9B5E,
0x21C66842, 0xF6E96C9A, 0x670C9C61, 0xABD388F0,
0x6A51A0D2, 0xD8542F68, 0x960FA728, 0xAB5133A3,
0x6EEF0B6C, 0x137A3BE4, 0xBA3BF050, 0x7EFB2A98,
0xA1F1651D, 0x39AF0176, 0x66CA593E, 0x82430E88,
0x8CEE8619, 0x456F9FB4, 0x7D84A5C3, 0x3B8B5EBE,
```

0xE06F75D8, 0x85C12073, 0x401A449F, 0x56C16AA6,
0x4ED3AA62, 0x363F7706, 0x1BFEDF72, 0x429B023D,
0x37D0D724, 0xD00A1248, 0xDB0FEAD3, 0x49F1C09B,
0x075372C9, 0x80991B7B, 0x25D479D8, 0xF6E8DEF7,
0xE3FE501A, 0xB6794C3B, 0x976CE0BD, 0x04C006BA,
0xC1A94FB6, 0x409F60C4, 0x5E5C9EC2, 0x196A2463,
0x68FB6FAF, 0x3E6C53B5, 0x1339B2EB, 0x3B52EC6F,
0x6DFC511F, 0x9B30952C, 0xCC814544, 0xAF5EBD09,
0xBEE3D004, 0xDE334AFD, 0x660F2807, 0x192E4BB3,
0xC0CBA857, 0x45C8740F, 0xD20B5F39, 0xB9D3FBDB,
0x5579C0BD, 0x1A60320A, 0xD6A100C6, 0x402C7279,
0x679F25FE, 0xFB1FA3CC, 0x8EA5E9F8, 0xDB3222F8,
0x3C7516DF, 0xFD616B15, 0x2F501EC8, 0xAD0552AB,
0x323DB5FA, 0xFD238760, 0x53317B48, 0x3E00DF82,
0x9E5C57BB, 0xCA6F8CA0, 0x1A87562E, 0xDF1769DB,
0xD542A8F6, 0x287EFFC3, 0xAC6732C6, 0x8C4F5573,
0x695B27B0, 0xBBCA58C8, 0xE1FFA35D, 0xB8F011A0,
0x10FA3D98, 0xFD2183B8, 0x4AFCB56C, 0x2DD1D35B,
0x9A53E479, 0xB6F84565, 0xD28E49BC, 0x4BFB9790,
0xE1DDF2DA, 0xA4CB7E33, 0x62FB1341, 0xCEE4C6E8,
0xEF20CADA, 0x36774C01, 0xD07E9EFE, 0x2BF11FB4,
0x95DBDA4D, 0xAE909198, 0xEAAD8E71, 0x6B93D5A0,
0xD08ED1D0, 0xAFC725E0, 0x8E3C5B2F, 0x8E7594B7,
0x8FF6E2FB, 0xF2122B64, 0x8888B812, 0x900DF01C,
0x4FAD5EA0, 0x688FC31C, 0xD1CFF191, 0xB3A8C1AD,
0x2F2F2218, 0xBE0E1777, 0xEA752DFE, 0x8B021FA1,
0xE5A0CC0F, 0xB56F74E8, 0x18ACF3D6, 0xCE89E299,
0xB4A84FE0, 0xFD13E0B7, 0x7CC43B81, 0xD2ADA8D9,
0x165FA266, 0x80957705, 0x93CC7314, 0x211A1477,
0xE6AD2065, 0x77B5FA86, 0xC75442F5, 0xFB9D35CF,
0xEBCDAF0C, 0x7B3E89A0, 0xD6411BD3, 0xAE1E7E49,
0x00250E2D, 0x2071B35E, 0x226800BB, 0x57B8E0AF,
0x2464369B, 0xF009B91E, 0x5563911D, 0x59DFA6AA,
0x78C14389, 0xD95A537F, 0x207D5BA2, 0x02E5B9C5,
0x83260376, 0x6295CFA9, 0x11C81968, 0x4E734A41,
0xB3472DCA, 0x7B14A94A, 0x1B510052, 0x9A532915,
0xD60F573F, 0xBC9BC6E4, 0x2B60A476, 0x81E67400,
0x08BA6FB5, 0x571BE91F, 0xF296EC6B, 0x2A0DD915,
0xB6636521, 0xE7B9F9B6, 0xFF34052E, 0xC5855664,
0x53B02D5D, 0xA99F8FA1, 0x08BA4799,

0x6E85076A],[0x4B7A70E9, 0xB5B32944, 0xDB75092E,
0xC4192623, 0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8,
0x8FEDB266, 0xECAA8C71, 0x699A17FF, 0x5664526C,
0xC2B19EE1, 0x193602A5, 0x75094C29, 0xA0591340,
0xE4183A3E, 0x3F54989A, 0x5B429D65, 0x6B8FE4D6,
0x99F73FD6, 0xA1D29C07, 0xEFE830F5, 0x4D2D38E6,
0xF0255DC1, 0x4CDD2086, 0x8470EB26, 0x6382E9C6,
0x021ECC5E, 0x09686B3F, 0x3EBAEFC9, 0x3C971814,
0x6B6A70A1, 0x687F3584, 0x52A0E286, 0xB79C5305,
0xAA500737, 0x3E07841C, 0x7FDEAE5C, 0x8E7D44EC,
0x5716F2B8, 0xB03ADA37, 0xF0500C0D, 0xF01C1F04,
0x0200B3FF, 0xAE0CF51A, 0x3CB574B2, 0x25837A58,
0xDC0921BD, 0xD19113F9, 0x7CA92FF6, 0x94324773,
0x22F54701, 0x3AE5E581, 0x37C2DADC, 0xC8B57634,
0x9AF3DDA7, 0xA9446146, 0x0FD0030E, 0xECC8C73E,
0xA4751E41, 0xE238CD99, 0x3BEA0E2F, 0x3280BBA1,
0x183EB331, 0x4E548B38, 0x4F6DB908, 0x6F420D03,
0xF60A04BF, 0x2CB81290, 0x24977C79, 0x5679B072,
0xBCAF89AF, 0xDE9A771F, 0xD9930810, 0xB38BAE12,
0xDCCF3F2E, 0x5512721F, 0x2E6B7124, 0x501ADDE6,
0x9F84CD87, 0x7A584718, 0x7408DA17, 0xBC9F9ABC,
0xE94B7D8C, 0xEC7AEC3A, 0xDB851DFA, 0x63094366,
0xC464C3D2, 0xEF1C1847, 0x3215D908, 0xDD433B37,
0x24C2BA16, 0x12A14D43, 0x2A65C451, 0x50940002,
0x133AE4DD, 0x71DFF89E, 0x10314E55, 0x81AC77D6,
0x5F11199B, 0x043556F1, 0xD7A3C76B, 0x3C11183B,
0x5924A509, 0xF28FE6ED, 0x97F1FBFA, 0x9EBABF2C,
0x1E153C6E, 0x86E34570, 0xEAE96FB1, 0x860E5E0A,
0x5A3E2AB3, 0x771FE71C, 0x4E3D06FA, 0x2965DCB9,
0x99E71D0F, 0x803E89D6, 0x5266C825, 0x2E4CC978,
0x9C10B36A, 0xC6150EBA, 0x94E2EA78, 0xA5FC3C53,
0x1E0A2DF4, 0xF2F74EA7, 0x361D2B3D, 0x1939260F,
0x19C27960, 0x5223A708, 0xF71312B6, 0xEBADFE6E,
0xEAC31F66, 0xE3BC4595, 0xA67BC883, 0xB17F37D1,
0x018CFF28, 0xC332DDEF, 0xBE6C5AA5, 0x65582185,
0x68AB9802, 0xEECEA50F, 0xDB2F953B, 0x2AEF7DAD,
0x5B6E2F84, 0x1521B628, 0x29076170, 0xECDD4775,
0x619F1510, 0x13CCA830, 0xEB61BD96, 0x0334FE1E,
0xAA0363CF, 0xB5735C90, 0x4C70A239, 0xD59E9E0B,
0xCBAADE14, 0xEECC86BC, 0x60622CA7, 0x9CAB5CAB,

0xB2F3846E, 0x648B1EAF, 0x19BDF0CA, 0xA02369B9,
0x655ABB50, 0x40685A32, 0x3C2AB4B3, 0x319EE9D5,
0xC021B8F7, 0x9B540B19, 0x875FA099, 0x95F7997E,
0x623D7DA8, 0xF837889A, 0x97E32D77, 0x11ED935F,
0x16681281, 0x0E358829, 0xC7E61FD6, 0x96DEDFA1,
0x7858BA99, 0x57F584A5, 0x1B227263, 0x9B83C3FF,
0x1AC24696, 0xCDB30AEB, 0x532E3054, 0x8FD948E4,
0x6DBC3128, 0x58EBF2EF, 0x34C6FFEA, 0xFE28ED61,
0xEE7C3C73, 0x5D4A14D9, 0xE864B7E3, 0x42105D14,
0x203E13E0, 0x45EEE2B6, 0xA3AAABEA, 0xDB6C4F15,
0xFACB4FD0, 0xC742F442, 0xEF6ABBB5, 0x654F3B1D,
0x41CD2105, 0xD81E799E, 0x86854DC7, 0xE44B476A,
0x3D816250, 0xCF62A1F2, 0x5B8D2646, 0xFC8883A0,
0xC1C7B6A3, 0x7F1524C3, 0x69CB7492, 0x47848A0B,
0x5692B285, 0x095BBF00, 0xAD19489D, 0x1462B174,
0x23820E00, 0x58428D2A, 0x0C55F5EA, 0x1DADF43E,
0x233F7061, 0x3372F092, 0x8D937E41, 0xD65FECF1,
0x6C223BDB, 0x7CDE3759, 0xCBEE7460, 0x4085F2A7,
0xCE77326E, 0xA6078084, 0x19F8509E, 0xE8EFD855,
0x61D99735, 0xA969A7AA, 0xC50C06C2, 0x5A04ABFC,
0x800BCADC, 0x9E447A2E, 0xC3453484, 0xFDD56705,
0x0E1E9EC9, 0xDB73DBD3, 0x105588CD, 0x675FDA79,
0xE3674340, 0xC5C43465, 0x713E38D8, 0x3D28F89E,
0xF16DFF20, 0x153E21E7, 0x8FB03D4A, 0xE6E39F2B,
0xDB83ADF7],[0xE93D5A68, 0x948140F7, 0xF64C261C,
0x94692934, 0x411520F7, 0x7602D4F7, 0xBCF46B2E,
0xD4A20068, 0xD4082471, 0x3320F46A, 0x43B7D4B7,
0x500061AF, 0x1E39F62E, 0x97244546, 0x14214F74,
0xBF8B8840, 0x4D95FC1D, 0x96B591AF, 0x70F4DDD3,
0x66A02F45, 0xBFBC09EC, 0x03BD9785, 0x7FAC6DD0,
0x31CB8504, 0x96EB27B3, 0x55FD3941, 0xDA2547E6,
0xABCA0A9A, 0x28507825, 0x530429F4, 0x0A2C86DA,
0xE9B66DFB, 0x68DC1462, 0xD7486900, 0x680EC0A4,
0x27A18DEE, 0x4F3FFEA2, 0xE887AD8C, 0xB58CE006,
0x7AF4D6B6, 0xAACE1E7C, 0xD3375FEC, 0xCE78A399,
0x406B2A42, 0x20FE9E35, 0xD9F385B9, 0xEE39D7AB,
0x3B124E8B, 0x1DC9FAF7, 0x4B6D1856, 0x26A36631,
0xEAE397B2, 0x3A6EFA74, 0xDD5B4332, 0x6841E7F7,
0xCA7820FB, 0xFB0AF54E, 0xD8FEB397, 0x454056AC,
0xBA489527, 0x55533A3A, 0x20838D87, 0xFE6BA9B7,

0xD096954B, 0x55A867BC, 0xA1159A58, 0xCCA92963,
0x99E1DB33, 0xA62A4A56, 0x3F3125F9, 0x5EF47E1C,
0x9029317C, 0xFDF8E802, 0x04272F70, 0x80BB155C,
0x05282CE3, 0x95C11548, 0xE4C66D22, 0x48C1133F,
0xC70F86DC, 0x07F9C9EE, 0x41041F0F, 0x404779A4,
0x5D886E17, 0x325F51EB, 0xD59BC0D1, 0xF2BCC18F,
0x41113564, 0x257B7834, 0x602A9C60, 0xDFF8E8A3,
0x1F636C1B, 0x0E12B4C2, 0x02E1329E, 0xAF664FD1,
0xCAD18115, 0x6B2395E0, 0x333E92E1, 0x3B240B62,
0xEEBEB922, 0x85B2A20E, 0xE6BA0D99, 0xDE720C8C,
0x2DA2F728, 0xD0127845, 0x95B794FD, 0x647D0862,
0xE7CCF5F0, 0x5449A36F, 0x877D48FA, 0xC39DFD27,
0xF33E8D1E, 0x0A476341, 0x992EFF74, 0x3A6F6EAB,
0xF4F8FD37, 0xA812DC60, 0xA1EBDDF8, 0x991BE14C,
0xDB6E6B0D, 0xC67B5510, 0x6D672C37, 0x2765D43B,
0xDCD0E804, 0xF1290DC7, 0xCC00FFA3, 0xB5390F92,
0x690FED0B, 0x667B9FFB, 0xCEDB7D9C, 0xA091CF0B,
0xD9155EA3, 0xBB132F88, 0x515BAD24, 0x7B9479BF,
0x763BD6EB, 0x37392EB3, 0xCC115979, 0x8026E297,
0xF42E312D, 0x6842ADA7, 0xC66A2B3B, 0x12754CCC,
0x782EF11C, 0x6A124237, 0xB79251E7, 0x06A1BBE6,
0x4BFB6350, 0x1A6B1018, 0x11CAEDFA, 0x3D25BDD8,
0xE2E1C3C9, 0x44421659, 0x0A121386, 0xD90CEC6E,
0xD5ABEA2A, 0x64AF674E, 0xDA86A85F, 0xBEBFE988,
0x64E4C3FE, 0x9DBC8057, 0xF0F7C086, 0x60787BF8,
0x6003604D, 0xD1FD8346, 0xF6381FB0, 0x7745AE04,
0xD736FCCC, 0x83426B33, 0xF01EAB71, 0xB0804187,
0x3C005E5F, 0x77A057BE, 0xBDE8AE24, 0x55464299,
0xBF582E61, 0x4E58F48F, 0xF2DDFDA2, 0xF474EF38,
0x8789BDC2, 0x5366F9C3, 0xC8B38E74, 0xB475F255,
0x46FCD9B9, 0x7AEB2661, 0x8B1DDF84, 0x846A0E79,
0x915F95E2, 0x466E598E, 0x20B45770, 0x8CD55591,
0xC902DE4C, 0xB90BACE1, 0xBB8205D0, 0x11A86248,
0x7574A99E, 0xB77F19B6, 0xE0A9DC09, 0x662D09A1,
0xC4324633, 0xE85A1F02, 0x09F0BE8C, 0x4A99A025,
0x1D6EFE10, 0x1AB93D1D, 0x0BA5A4DF, 0xA186F20F,
0x2868F169, 0xDCB7DA83, 0x573906FE, 0xA1E2CE9B,
0x4FCD7F52, 0x50115E01, 0xA70683FA, 0xA002B5C4,
0x0DE6D027, 0x9AF88C27, 0x773F8641, 0xC3604C06,
0x61A806B5, 0xF0177A28, 0xC0F586E0, 0x006058AA,

0x30DC7D62, 0x11E69ED7, 0x2338EA63, 0x53C2DD94,
0xC2C21634, 0xBBCBEE56, 0x90BCB6DE, 0xEBFC7DA1,
0xCE591D76, 0x6F05E409, 0x4B7C0188, 0x39720A3D,
0x7C927C24, 0x86E3725F, 0x724D9DB9, 0x1AC15BB4,
0xD39EB8FC, 0xED545578, 0x08FCA5B5, 0xD83D7CD3,
0x4DAD0FC4, 0x1E50EF5E, 0xB161E6F8, 0xA28514D9,
0x6C51133C, 0x6FD5C7E7, 0x56E14EC4, 0x362ABFCE,
0xDDC6C837, 0xD79A3234, 0x92638212, 0x670EFA8E,
0x406000E0], [0x3A39CE37, 0xD3FAF5CF, 0xABC27737,
0x5AC52D1B, 0x5CB0679E, 0x4FA33742, 0xD3822740,
0x99BC9BBE, 0xD5118E9D, 0xBF0F7315, 0xD62D1C7E,
0xC700C47B, 0xB78C1B6B, 0x21A19045, 0xB26EB1BE,
0x6A366EB4, 0x5748AB2F, 0xBC946E79, 0xC6A376D2,
0x6549C2C8, 0x530FF8EE, 0x468DDE7D, 0xD5730A1D,
0x4CD04DC6, 0x2939BBDB, 0xA9BA4650, 0xAC9526E8,
0xBE5EE304, 0xA1FAD5F0, 0x6A2D519A, 0x63EF8CE2,
0x9A86EE22, 0xC089C2B8, 0x43242EF6, 0xA51E03AA,
0x9CF2D0A4, 0x83C061BA, 0x9BE96A4D, 0x8FE51550,
0xBA645BD6, 0x2826A2F9, 0xA73A3AE1, 0x4BA99586,
0xEF5562E9, 0xC72FEFD3, 0xF752F7DA, 0x3F046F69,
0x77FA0A59, 0x80E4A915, 0x87B08601, 0x9B09E6AD,
0x3B3EE593, 0xE990FD5A, 0x9E34D797, 0x2CF0B7D9,
0x022B8B51, 0x96D5AC3A, 0x017DA67D, 0xD1CF3ED6,
0x7C7D2D28, 0x1F9F25CF, 0xADF2B89B, 0x5AD6B472,
0x5A88F54C, 0xE029AC71, 0xE019A5E6, 0x47B0ACFD,
0xED93FA9B, 0xE8D3C48D, 0x283B57CC, 0xF8D56629,
0x79132E28, 0x785F0191, 0xED756055, 0xF7960E44,
0xE3D35E8C, 0x15056DD4, 0x88F46DBA, 0x03A16125,
0x0564F0BD, 0xC3EB9E15, 0x3C9057A2, 0x97271AEC,
0xA93A072A, 0x1B3F6D9B, 0x1E6321F5, 0xF59C66FB,
0x26DCF319, 0x7533D928, 0xB155FDF5, 0x03563482,
0x8ABA3CBB, 0x28517711, 0xC20AD9F8, 0xABCC5167,
0xCCAD925F, 0x4DE81751, 0x3830DC8E, 0x379D5862,
0x9320F991, 0xEA7A90C2, 0xFB3E7BCE, 0x5121CE64,
0x774FBE32, 0xA8B6E37E, 0xC3293D46, 0x48DE5369,
0x6413E680, 0xA2AE0810, 0xDD6DB224, 0x69852DFD,
0x09072166, 0xB39A460A, 0x6445C0DD, 0x586CDECF,
0x1C20C8AE, 0x5BBEF7DD, 0x1B588D40, 0xCCD2017F,
0x6BB4E3BB, 0xDDA26A7E, 0x3A59FF45, 0x3E350A44,
0xBCB4CDD5, 0x72EACEA8, 0xFA6484BB, 0x8D6612AE,

0xBF3C6F47, 0xD29BE463, 0x542F5D9E, 0xAEC2771B,
0xF64E6370, 0x740E0D8D, 0xE75B1357, 0xF8721671,
0xAF537D5D, 0x4040CB08, 0x4EB4E2CC, 0x34D2466A,
0x0115AF84, 0xE1B00428, 0x95983A1D, 0x06B89FB4,
0xCE6EA048, 0x6F3F3B82, 0x3520AB82, 0x011A1D4B,
0x277227F8, 0x611560B1, 0xE7933FDC, 0xBB3A792B,
0x344525BD, 0xA08839E1, 0x51CE794B, 0x2F32C9B7,
0xA01FBAC9, 0xE01CC87E, 0xBCC7D1F6, 0xCF0111C3,
0xA1E8AAC7, 0x1A908749, 0xD44FBD9A, 0xD0DADECB,
0xD50ADA38, 0x0339C32A, 0xC6913667, 0x8DF9317C,
0xE0B12B4F, 0xF79E59B7, 0x43F5BB3A, 0xF2D519FF,
0x27D9459C, 0xBF97222C, 0x15E6FC2A, 0x0F91FC71,
0x9B941525, 0xFAE59361, 0xCEB69CEB, 0xC2A86459,
0x12BAA8D1, 0xB6C1075E, 0xE3056A0C, 0x10D25065,
0xCB03A442, 0xE0EC6E0E, 0x1698DB3B, 0x4C98A0BE,
0x3278E964, 0x9F1F9532, 0xE0D392DF, 0xD3A0342B,
0x8971F21E, 0x1B0A7441, 0x4BA3348C, 0xC5BE7120,
0xC37632D8, 0xDF359F8D, 0x9B992F2E, 0xE60B6F47,
0x0FE3F11D, 0xE54CDA54, 0x1EDAD891, 0xCE6279CF,
0xCD3E7E6F, 0x1618B166, 0xFD2C1D05, 0x848FD2C5,
0xF6FB2299, 0xF523F357, 0xA6327623, 0x93A83531,
0x56CCCD02, 0xACF08162, 0x5A75EBB5, 0x6E163697,
0x88D273CC, 0xDE966292, 0x81B949D0, 0x4C50901B,
0x71C65614, 0xE6C6C7BD, 0x327A140A, 0x45E1D006,
0xC3F27B9A, 0xC9AA53FD, 0x62A80F00, 0xBB25BFE2,
0x35BDD2F6, 0x71126905, 0xB2040222, 0xB6CBCF7C,
0xCD769C2B, 0x53113EC0, 0x1640E3D3, 0x38ABBD60,
0x2547ADF0, 0xBA38209C, 0xF746CE76, 0x77AFA1C5,
0x20756060, 0x85CBFE4E, 0x8AE88DD8, 0x7AAAF9B0,
0x4CF9AA7E, 0x1948C25C, 0x02FB8A8C, 0x01C36AE4,
0xD6EBE1F9, 0x90D4F869, 0xA65CDEA0, 0x3F09252D,
0xC208E69F, 0xB74E6132, 0xCE77E25B, 0x578FDFE3,
0x3AC372E6]]
key = [0x4B7A70E9, 0xB5B32944, 0xDB75092E, 0xC4192623,
0xAD6EA6B0, 0x49A7DF7D, 0x9CEE60B8, 0x8FEDB266,
0xECAA8C71, 0x699A17FF, 0x5664526C, 0xC2B19EE1,
0x193602A5, 0x75094C29]
p_new = p.copy()
def swap(a,b):
    temp = a

```
    a = b
    b = temp
    return a,b
def driver():
        for i in range(0,18):
                p[i] = p[i]^key[i%14]
        k = 0
        data = 0
        for i in range(0,9):
            temp = encryption(data)
            p[k] = temp >> 32
            k+=1
            p[k] = temp & 0xffffffff
            k+=1
            data = temp
        encrypt_data = int(input("Enter data to encrypt: "))
        encrypted_data = encryption(encrypt_data)
        print("Encrypted data : ",encrypted_data)
        decrypted_data = decryption(encrypted_data)
        print("Decrypted data : ",decrypted_data)
def encryption(data):
        L = data>>32
        R = data & 0xffffffff
        for i in range(0,16):
                L = p[i]^L
                L1 = func(L)
                R = R^func(L1)
                L,R = swap(L,R)
        L,R = swap(L,R)
        L = L^p[17]
        R = R^p[16]
        encrypted = (L<<32) ^ R
        return encrypted
def func(L):
    temp = s[0][L >> 24]
    temp = (temp + s[1][L >> 16 & 0xff]) % 2**32
    temp = temp ^ s[2][L >> 8 & 0xff]
    temp = (temp + s[3][L & 0xff]) % 2**32
    return temp
def decryption(data):
```

```
    L = data >> 32
    R = data & 0xffffffff
    for i in range(17, 1, -1):
        L = p[i]^L
        L1 = func(L)
        R = R^func(L1)
        L,R = swap(L,R)
    L,R = swap(L,R)
    L = L^p[0]
    R = R^p[1]
    decrypted_data1 = (L<<32) ^ R
    return decrypted_data1
driver()
```

OUTPUT:

```
Enter data to encrypt: 672633
Encrypted data :  17167022931085337131
Decrypted data :  672633
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using Blowfish algorithm.

AIM: Write a program to implement the AES algorithm logic.

PROGRAM:

```
Sbox = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01,
0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD,
0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5,
0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12,
0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B,
0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A,
0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45,
0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6,
0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4,
0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE,
0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3,
0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C,
0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8,
0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35,
0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E,
0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16,
)
InvSbox = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40,
0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
```

```
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E,
0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE,
0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B,
0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4,
0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E,
0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7,
0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1,
0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2,
0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9,
0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7,
0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A,
0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1,
0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D,
0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8,
0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1,
0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,
)
xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a
<< 1)
Rcon = (
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
    0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
    0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
)
def text2matrix(text):
    matrix = []
```

```
    for i in range(16):
        byte = (text >> (8 * (15 - i))) & 0xFF
        if i % 4 == 0:
            matrix.append([byte])
        else:
            matrix[i//4].append(byte)
    return matrix
def matrix2text(matrix):
    text = 0
    for i in range(4):
        for j in range(4):
            text |= (matrix[i][j] << (120 - 8 * (4 * i + j)))
    return text
class AES:
    def __init__(self, master_key):
        self.change_key(master_key)
    def change_key(self, master_key):
        self.round_keys = text2matrix(master_key)
        for i in range(4, 4 * 11):
            self.round_keys.append([])
            if i % 4 == 0:
                byte = self.round_keys[i - 4][0]^ Sbox[self.round_keys[i
- 1][1]]^ Rcon[i // 4]
                self.round_keys[i].append(byte)
                for j in range(1, 4):
                    byte = self.round_keys[i - 4][j]^
Sbox[self.round_keys[i - 1][(j + 1) % 4]]
                    self.round_keys[i].append(byte)
            else:
                for j in range(4):
                    byte = self.round_keys[i - 4][j]^ self.round_keys[i -
1][j]
                    self.round_keys[i].append(byte)
    def encrypt(self, plaintext):
        self.plain_state = text2matrix(plaintext)
        self.__add_round_key(self.plain_state, self.round_keys[:4])
        for i in range(1, 10):
            self.__round_encrypt(self.plain_state, self.round_keys[4 * i
: 4 * (i + 1)])
        self.__sub_bytes(self.plain_state)
        self.__shift_rows(self.plain_state)
```

```
        self.__add_round_key(self.plain_state, self.round_keys[40:])
        return matrix2text(self.plain_state)
    def decrypt(self, ciphertext):
        self.cipher_state = text2matrix(ciphertext)
        self.__add_round_key(self.cipher_state, self.round_keys[40:])
        self.__inv_shift_rows(self.cipher_state)
        self.__inv_sub_bytes(self.cipher_state)
        for i in range(9, 0, -1):
            self.__round_decrypt(self.cipher_state, self.round_keys[4 *
i : 4 * (i + 1)])
        self.__add_round_key(self.cipher_state, self.round_keys[:4])
        return matrix2text(self.cipher_state)
    def __add_round_key(self, s, k):
        for i in range(4):
            for j in range(4):
                s[i][j] ^= k[i][j]
    def __round_encrypt(self, state_matrix, key_matrix):
        self.__sub_bytes(state_matrix)
        self.__shift_rows(state_matrix)
        self.__mix_columns(state_matrix)
        self.__add_round_key(state_matrix, key_matrix)
    def __round_decrypt(self, state_matrix, key_matrix):
        self.__add_round_key(state_matrix, key_matrix)
        self.__inv_mix_columns(state_matrix)
        self.__inv_shift_rows(state_matrix)
        self.__inv_sub_bytes(state_matrix)
    def __sub_bytes(self, s):
        for i in range(4):
            for j in range(4):
                s[i][j] = Sbox[s[i][j]]
    def __inv_sub_bytes(self, s):
        for i in range(4):
            for j in range(4):
                s[i][j] = InvSbox[s[i][j]]
    def __shift_rows(self, s):
        s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
        s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
        s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]
    def __inv_shift_rows(self, s):
        s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]
        s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
```

```
        s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]
    def __mix_single_column(self, a):
        t = a[0] ^ a[1] ^ a[2] ^ a[3]
        u = a[0]
        a[0] ^= t ^ xtime(a[0] ^ a[1])
        a[1] ^= t ^ xtime(a[1] ^ a[2])
        a[2] ^= t ^ xtime(a[2] ^ a[3])
        a[3] ^= t ^ xtime(a[3] ^ u)
    def __mix_columns(self, s):
        for i in range(4):
            self.__mix_single_column(s[i])
    def __inv_mix_columns(self, s):
        for i in range(4):
            u = xtime(xtime(s[i][0] ^ s[i][2]))
            v = xtime(xtime(s[i][1] ^ s[i][3]))
            s[i][0] ^= u
            s[i][1] ^= v
            s[i][2] ^= u
            s[i][3] ^= v
        self.__mix_columns(s)
master_key = 0x2b7e151628aed2a6abf7158809cf4f3c
aes=AES(master_key)
plaintext=int(input("Enter the plain text: "))
encrypted = aes.encrypt(plaintext)
print("Cipher text:",encrypted)
decrypted = aes.decrypt(encrypted)
print("Decrypted text:",decrypted)
```

OUTPUT:

```
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python AES.py
Enter the plain text: 412425234324
cipher text: 159121485027414208789628474209443426258
decrypted text: 412425234324
PS D:\Engineering\SEM-7\Cryptography and Network Security\Lab Work\Programs> python AES.py
Enter the plain text: 42124325324
Cipher text: 91064687550319156027004710911803535229
Decrypted text: 42124325324
```

AIM: Write a program to implement the RC4 algorithm logic.

DESCRIPTION: In this program, we are trying to perform encryption and decryption of the given string using RC4 algorithm.

ALGORITHM:

Step-1: Start
Step-2: User Input: The program takes two inputs from the user - a plaintext message (ptext) and an encryption key (key).
Step-3: Key Expansion: The key is expanded into an array s of 256 bytes and an array k of the same length as the plaintext message. This is done using a key-scheduling algorithm (KSA).
Step-4: Initialization: Two indices i and j are initialized to 0. A temporary variable temp is also initialized.
Step-5: Key Mixing: The algorithm mixes the bytes in arrays s and k to create a permutation in s. This is done in a loop, and the values of i, j, and temp are updated during each iteration.
Step-6: Encryption: For each byte in the plaintext message, the algorithm generates a pseudorandom byte z from the s array. Then, it computes the ciphertext byte by XORing z with the corresponding byte from the plaintext message and stores it in the cipher array.
Step-7: Decryption: To decrypt, the same pseudorandom byte z is generated, but this time, it's XORed with the ciphertext byte to recover the original plaintext byte. The decrypted bytes are stored in the decrypt array.
Step-8: Display: Finally, the encrypted and decrypted messages are displayed using the display function, which converts the integer values to characters and prints them.
Step-9: End

PROGRAM:

```
#include <stdio.h>
#include <string.h>
void display(int disp[], int length) {
        int l;
    char convert[length];
    for (l = 0; l < length; l++) {
        convert[l] = (char)disp[l];
        printf("%c", convert[l]);
    }
}
int main() {
    int temp = 0, i, l;
    char ptext[256];
    char key[256];
    int s[256];
    int k[256];
    printf("\nENTER PLAIN TEXT: ");
    fgets(ptext, sizeof(ptext), stdin);
    printf("ENTER KEY TEXT: ");
    fgets(key, sizeof(key), stdin);
    int ptextLen = strlen(ptext) - 1;
    int keyLen = strlen(key) - 1;
    int cipher[ptextLen];
    int decrypt[ptextLen];
    int ptexti[ptextLen];
    int keyi[keyLen];
    for (i = 0; i < ptextLen; i++) {
        ptexti[i] = (int)ptext[i];
    }
    for (i = 0; i < keyLen; i++) {
        keyi[i] = (int)key[i];
    }
    for (i = 0; i < 255; i++) {
        s[i] = i;
        k[i] = keyi[i % keyLen];
    }
    int j = 0;
    for (i = 0; i < 255; i++) {
        j = (j + s[i] + k[i]) % 256;
```

```
        temp = s[i];
        s[i] = s[j];
        s[j] = temp;
    }
    i = 0;
    j = 0;
    int z = 0;
    for (l = 0; l < ptextLen; l++) {
        i = (l + 1) % 256;
        j = (j + s[i]) % 256;
        temp = s[i];
        s[i] = s[j];
        s[j] = temp;
        z = s[(s[i] + s[j]) % 256];
        cipher[l] = z ^ ptexti[l];
        decrypt[l] = z ^ cipher[l];
    }
    printf("ENCRYPTED: ");
    display(cipher, ptextLen);
    printf("\nDECRYPTED: ");
    display(decrypt, ptextLen);
    return 0;
}
```

OUTPUT:

```
ENTER PLAIN TEXT: cryptography
ENTER KEY TEXT: security
ENCRYPTED: □d •δ/Åo»ù²Y
DECRYPTED: cryptography
--------------------------------
Process exited after 8.587 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

By executing the above, we have successfully performed encryption and decryption for the given string using RC4 algorithm.