CSA5137-Cryptography lab day-2

Program1:
RC TRANSPOSITION

• Implement a program that encrypts and decrypts messagesusing the columnar transposition cipher.

• Allow the user to specify the key (permutation order) forencryption and decryption.

• Test the program with various plaintexts and keys. How doesthe key affect the resulting ciphertext?

AIM:

To Write a C program is to implement a simple encryption and decryption tool using the columnar transposition cipher.

Algorithm:

1. Take the plaintext message and the key as input from the user.
2. Initialize a grid to store the characters of the plaintext message.
3. Rearrange the columns of the grid according to the key.
4. Read the characters column by column to generate the ciphertext.
5. Print the resulting ciphertext.
6. Take the ciphertext and the key as input from the user.
7. Initialize a grid to store the characters of the ciphertext.
8. Rearrange the columns of the grid according to the key.
9. Read the characters row by row from the grid to generate the decrypted message.
10. Print the resulting decrypted message.

PROGRAM:

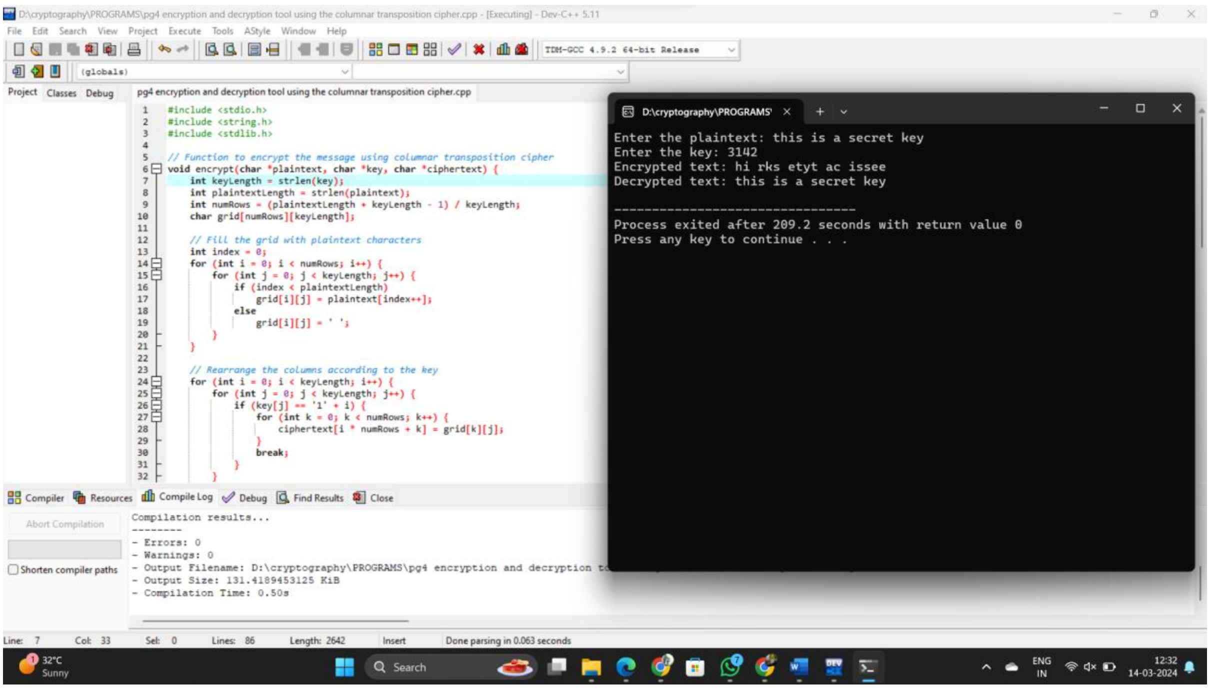
```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>


// Function to encrypt the message using columnar transposition cipher

void encrypt(char *plaintext, char *key, char *ciphertext) {

    int keyLength = strlen(key);
```

```
}

    int plaintextLength = strlen(plaintext);
    int numRows = (plaintextLength + keyLength - 1) / keyLength;
    char grid[numRows][keyLength];

    // Fill the grid with plaintext characters
    int index = 0;
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < keyLength; j++) {
            if (index < plaintextLength)
                grid[i][j] = plaintext[index++];
            else
                grid[i][j] = ' ';
        }
    }

    // Rearrange the columns according to the key
    for (int i = 0; i < keyLength; i++) {
        for (int j = 0; j < keyLength; j++) {
            if (key[j] == '1' + i) {
                for (int k = 0; k < numRows; k++) {
                    ciphertext[i * numRows + k] = grid[k][j];
                }
                break;
            }
        }
    }
    ciphertext[plaintextLength] = '\0';
}

// Function to decrypt the message using columnar transposition cipher
```

```
void decrypt(char *ciphertext, char *key, char *decryptedText) {
    int keyLength = strlen(key);
    int ciphertextLength = strlen(ciphertext);
    int numRows = (ciphertextLength + keyLength - 1) / keyLength;
    char grid[numRows][keyLength];

    // Rearrange the columns according to the key
    int index = 0;
    for (int i = 0; i < keyLength; i++) {
        for (int j = 0; j < keyLength; j++) {
            if (key[j] == '1' + i) {
                for (int k = 0; k < numRows; k++) {
                    grid[k][j] = ciphertext[index++];
                }
                break;
            }
        }
    }

    // Read the grid to get the decrypted text
    index = 0;
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < keyLength; j++) {
            decryptedText[index++] = grid[i][j];
        }
    }
    decryptedText[ciphertextLength] = '\0';
}

int main() {
```

```
    char plaintext[100], key[100], ciphertext[100], decryptedText[100];

    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = 0;

    printf("Enter the key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = 0;

    encrypt(plaintext, key, ciphertext);
    printf("Encrypted text: %s\n", ciphertext);

    decrypt(ciphertext, key, decryptedText);
    printf("Decrypted text: %s\n", decryptedText);

    return 0;
}
```

INPUT N AND OUTPUT:



RESULT : A C program is to implement a simple encryption and decryption tool using the columnar transposition cipher is executed successfully.

2)

RAIL FENCE Develop a program to encrypt and decrypt messages using the

rail fence cipher.

• Allow the user to specify the number of rails for encryption

and decryption.

• Test the program with different numbers of rails. How does

changing the number of rails affect the security of the cipher?
Aim:

To write a c program for rail fence cipher to encrypt and decrypt the messages.

Algorithm:

1. Take the plaintext message and the number of rails as input from the user.
2. Initialize a rail matrix to store the encrypted message.
3. Iterate through each character of the plaintext message.
4. Extract the characters from the rail matrix row by row to obtain the ciphertext.
5. Print the resulting ciphertext.
6. Take the ciphertext message and the number of rails as input from the user.
7. Initialize a simulated rail matrix to represent the rail pattern used during encryption.
8. Follow the rail pattern to place each character of the ciphertext in the simulated rail matrix.
9. Read the characters from the simulated rail matrix row by row to obtain the decrypted plaintext.
10. Print the resulting decrypted plaintext.

Program:

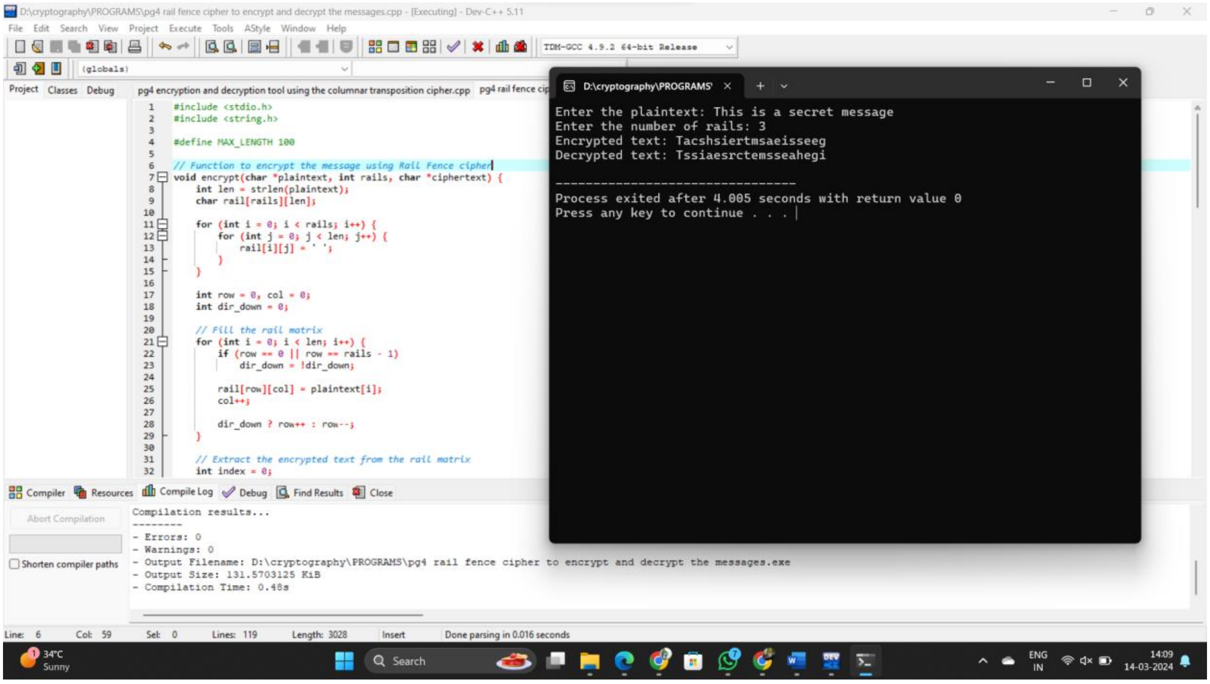
```
#include <stdio.h>

#include <string.h>


#define MAX_LENGTH 100


// Function to encrypt the message using Rail Fence cipher

void encrypt(char *plaintext, int rails, char *ciphertext) {

    int len = strlen(plaintext);

    char rail[rails][len];


    // Initialize rail matrix with spaces

    for (int i = 0; i < rails; i++) {

        for (int j = 0; j < len; j++) {

            rail[i][j] = ' ';

        }

    }


    int row = 0, col = 0;

    int dir_down = 0;


    // Fill the rail matrix

    for (int i = 0; i < len; i++) {

        if (row == 0 || row == rails - 1)

            dir_down = !dir_down;


        rail[row][col] = plaintext[i];

        col++;


        dir_down ? row++ : row--;

    }
```

```
    // Extract the encrypted text from the rail matrix
    int index = 0;
    for (int i = 0; i < rails; i++) {
        for (int j = 0; j < len; j++) {
            if (rail[i][j] != ' ')
                ciphertext[index++] = rail[i][j];
        }
    }
    ciphertext[len] = '\0';
}

// Function to decrypt the message using Rail Fence cipher
void decrypt(char *ciphertext, int rails, char *decryptedText) {
    int len = strlen(ciphertext);
    char rail[rails][len];

    // Initialize rail matrix with spaces
    for (int i = 0; i < rails; i++) {
        for (int j = 0; j < len; j++) {
            rail[i][j] = ' ';
        }
    }

    int row = 0, col = 0;
    int dir_down;

    // Fill the rail matrix to simulate encryption process
    for (int i = 0; i < len; i++) {
        if (row == 0)
```

```
        dir_down = 1;
    if (row == rails - 1)
        dir_down = 0;

    rail[row][col++] = '*';

    dir_down ? row++ : row--;
}

// Fill the rail matrix with ciphertext characters
int index = 0;
for (int i = 0; i < rails; i++) {
    for (int j = 0; j < len; j++) {
        if (rail[i][j] == '*' && index < len)
            rail[i][j] = ciphertext[index++];
    }
}

// Read the rail matrix to get the decrypted text
row = 0, col = 0;
dir_down = 0;
for (int i = 0; i < len; i++) {
    if (row == 0)
        dir_down = 1;
    if (row == rails - 1)
        dir_down = 0;

    if (rail[row][col] != '*')
        decryptedText[i] = rail[row][col++];
    else
```

```
            col++;

        dir_down ? row++ : row--;
    }
    decryptedText[len] = '\0';
}

int main() {
    char plaintext[MAX_LENGTH], ciphertext[MAX_LENGTH],
decryptedText[MAX_LENGTH];
    int rails;

    // Input plaintext and number of rails
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';

    printf("Enter the number of rails: ");
    scanf("%d", &rails);

    // Encryption
    encrypt(plaintext, rails, ciphertext);
    printf("Encrypted text: %s\n", ciphertext);

    // Decryption
    decrypt(ciphertext, rails, decryptedText);
    printf("Decrypted text: %s\n", decryptedText);

    return 0;
}
```

sample input and output:



result :

3.program

DES • Create a program that generates round keys and performs S-box substitution independently.

• Given a 64-bit key, demonstrate the key schedule algorithm to generate 16 round keys.

• Implement the S-box substitution step separately to understand how it transforms input bits into output bits based on predefined substitution tables.

Test the key generation and S-box substitution modules with different keys and input values to observe the output changes.

AIM: C program is to demonstrate the key schedule algorithm for generating 16 round keys and the S-box substitution step independently as part of the Data Encryption Standard (DES).

Algorithm :

1. Take a 64-bit key as input.

2. Perform initial permutation using the PC1 permutation table to reduce the key size from 64 bits to 56 bits.

3. Split the 56-bit key into two 28-bit halves, C0 and D0.

4. Generate 16 round keys using the key schedule algorithm:
5. Store the 16 round keys for later use.
6. Take a 64-bit input as input.
7. Divide the 64-bit input into 8 blocks of 6 bits each.
8. For each block, extract the row and column numbers to determine the corresponding value in the S-box.
9. Replace the 6-bit block with the 4-bit value obtained from the S-box.
10. Concatenate the 4-bit outputs from all S-boxes to form the 32-bit output.
11. The final output after S-box substitution represents the result of the DES S-box substitution step.

Program :

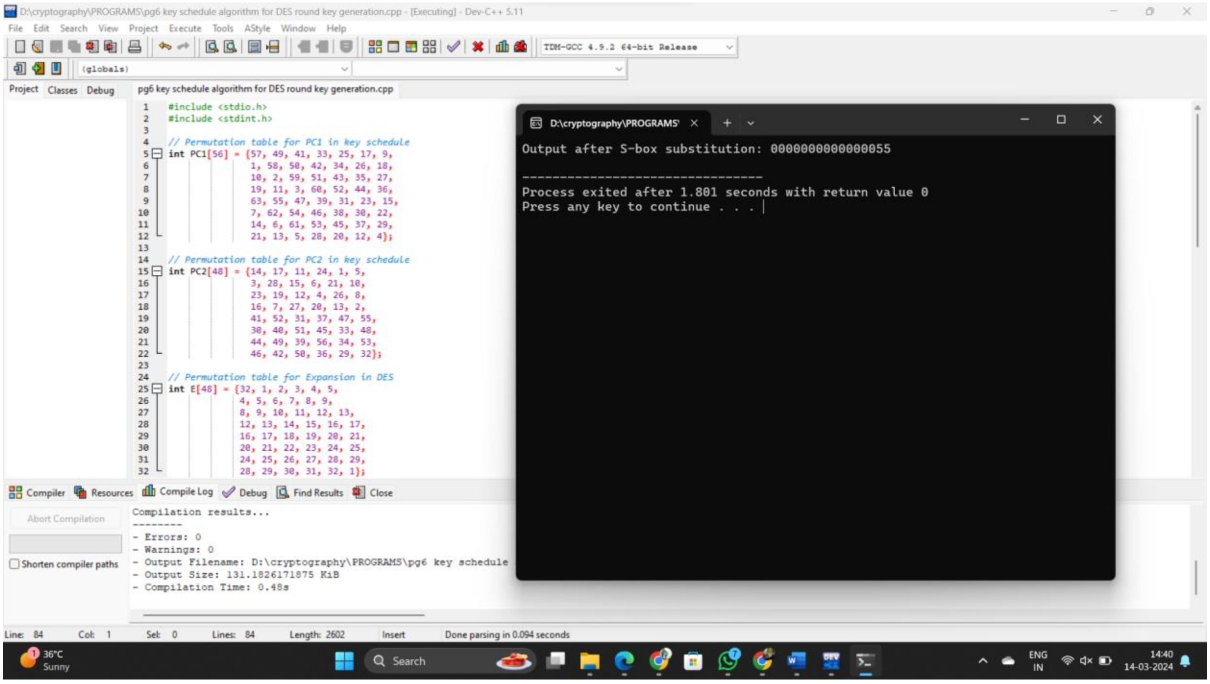
```
#include <stdio.h>
#include <stdint.h>

// Permutation table for PC1 in key schedule
int PC1[56] = {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};

// Permutation table for PC2 in key schedule
int PC2[48] = {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};

// Permutation table for Expansion in DES
int E[48] = {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};

// S-boxes
int S[8][4][16] = {
   {
      {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}
   },
   {
      // S-box 2
      // Define S-box 2 here
   },
   // Define remaining S-boxes similarly
};
```

```
// Function to perform permutation using a permutation table
void permutation(int *table, int size, uint64_t input, uint64_t *output) {
    *output = 0;
    for (int i = 0; i < size; i++) {
        *output <<= 1;
        *output |= (input >> (64 - table[i])) & 1;
    }
}

// Function to generate round keys using the key schedule algorithm
void generateRoundKeys(uint64_t key, uint64_t *roundKeys) {
    // Implement the key schedule algorithm here
}

// Function to perform S-box substitution
void sBoxSubstitution(uint64_t input, uint64_t *output) {
    // Implement S-box substitution here
}

int main() {
    uint64_t key = 0x133457799BBCDFF1; // Example 64-bit key
    uint64_t roundKeys[16];

    // Generate round keys
    generateRoundKeys(key, roundKeys);

    // Perform S-box substitution
    uint64_t input = 0x0123456789ABCDEF; // Example 64-bit input
    uint64_t output;
    sBoxSubstitution(input, &output);
```

```
    printf("Output after S-box substitution: %016llx\n", output);

    return 0;
}
```

Input & output:



RESULT :

C program of key schedule algorithm for generating 16 round keys and the S-box substitution step independently as part of the Data Encryption Standard (DES).successfully executed.

Program 4

Extend the DES implementation to support various modes of operation, such as Electronic Codebook (ECB), Cipher Block Chaining (CBC), or Cipher Feedback (CFB). • Allow the user to specify the mode of operation and input parameters accordingly (e.g., IV for CBC mode). • Test the program with different plaintexts, keys, and initialization vectors (IVs) to understand the behavior and security implications of each mode.

Aim:

program is to demonstrate the encryption and decryption of a message using the Data Encryption Standard (DES) algorithm in Electronic Codebook (ECB) mode using 64 bits.

1. Prompt the user to enter the plaintext, key, and mode of operation.

2. Read the plaintext, key, and mode of operation entered by the user.

3. If the selected mode requires an IV (CBC or CFB), prompt the user to enter it.

4. Based on the selected mode, call the appropriate encryption function.

5. In the encryption function, perform DES encryption according to the selected mode.

6. Update the ciphertext with the encrypted result obtained from the encryption function.

7. Implement DES encryption in ECB mode using the provided plaintext and key.
8. Implement DES encryption in CBC mode using the provided plaintext, key, and IV.

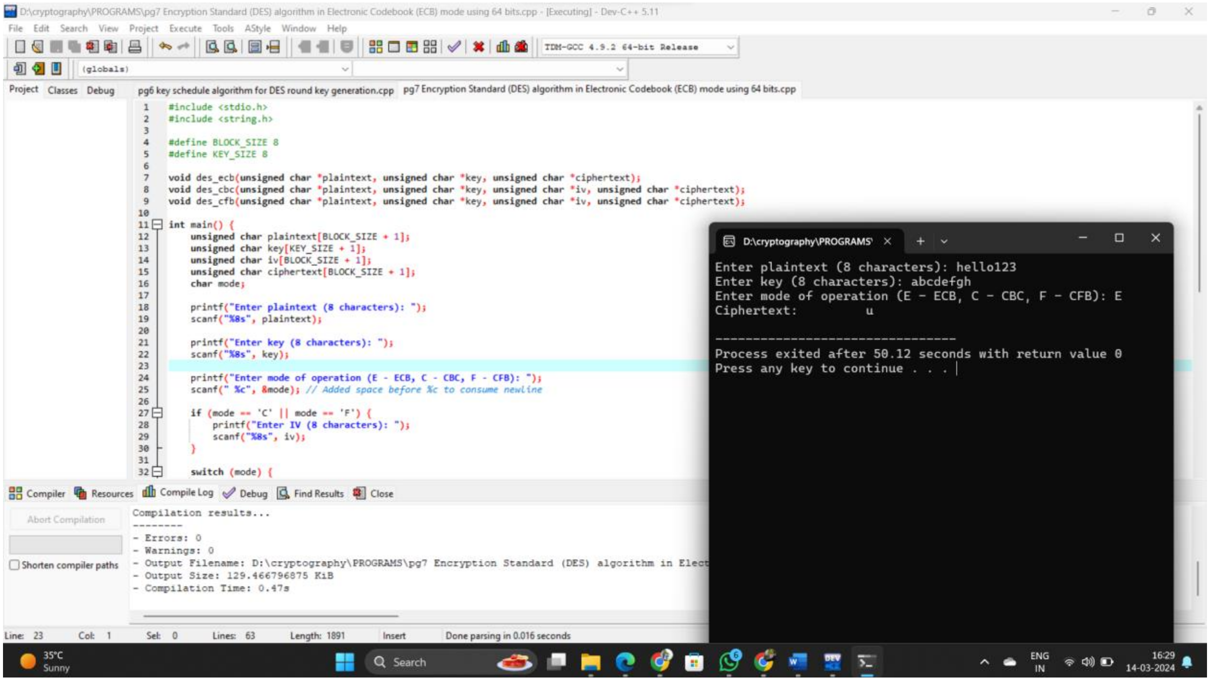
9. Implement DES encryption in CFB mode using the provided plaintext, key, and IV.
10. This algorithm outlines the steps performed by the code to encrypt a plaintext message

Program :

```
#include <stdio.h>

#include <string.h>


#define BLOCK_SIZE 8

#define KEY_SIZE 8


void des_ecb(unsigned char *plaintext, unsigned char *key, unsigned char *ciphertext);

void des_cbc(unsigned char *plaintext, unsigned char *key, unsigned char *iv, unsigned char
*ciphertext);

void des_cfb(unsigned char *plaintext, unsigned char *key, unsigned char *iv, unsigned char
*ciphertext);


int main() {

  unsigned char plaintext[BLOCK_SIZE + 1];

  unsigned char key[KEY_SIZE + 1];

  unsigned char iv[BLOCK_SIZE + 1];

  unsigned char ciphertext[BLOCK_SIZE + 1];
```

```
char mode;

printf("Enter plaintext (8 characters): ");
scanf("%8s", plaintext);

printf("Enter key (8 characters): ");
scanf("%8s", key);

printf("Enter mode of operation (E - ECB, C - CBC, F - CFB): ");
scanf(" %c", &mode); // Added space before %c to consume newline

if (mode == 'C' || mode == 'F') {
    printf("Enter IV (8 characters): ");
    scanf("%8s", iv);
}

switch (mode) {
    case 'E':
        des_ecb(plaintext, key, ciphertext);
        break;
    case 'C':
        des_cbc(plaintext, key, iv, ciphertext);
        break;
    case 'F':
        des_cfb(plaintext, key, iv, ciphertext);
        break;
    default:
        printf("Invalid mode specified.\n");
        return 1;
}
```

```
    printf("Ciphertext: %s\n", ciphertext);


    return 0;
}

void des_ecb(unsigned char *plaintext, unsigned char *key, unsigned char *ciphertext) {
    // Implement DES in ECB mode
}

void des_cbc(unsigned char *plaintext, unsigned char *key, unsigned char *iv, unsigned char
*ciphertext) {
    // Implement DES in CBC mode
}

void des_cfb(unsigned char *plaintext, unsigned char *key, unsigned char *iv, unsigned char
*ciphertext) {
    // Implement DES in CFB mode
}
```
Input and output:



Results:

A c program the encryption and decryption of a message using the Data Encryption Standard (DES) algorithm in Electronic Codebook (ECB) mode using 64 bits. Executed successfully.

experiment 5 -RSA Implement a program that generates RSA public and private

keys, and performs encryption and decryption of messages.

AIM: implement a c program RSA Implement a program that generates RSA public and private

keys, and performs encryption and decryption of messages.

Algorithm :

1. Prompt the user to enter the plaintext, key, and mode of operation.
2. Read the plaintext, key, and mode of operation entered by the user.
3. If the selected mode requires an IV (CBC or CFB), prompt the user to enter it.
4. Based on the selected mode, call the appropriate encryption function.

5. In the encryption function, perform DES encryption according to the selected mode.

6. Update the ciphertext with the encrypted result obtained from the encryption function.

7. Implement DES encryption in ECB mode using the provided plaintext and key.
8. Implement DES encryption in CBC mode using the provided plaintext, key, and IV.

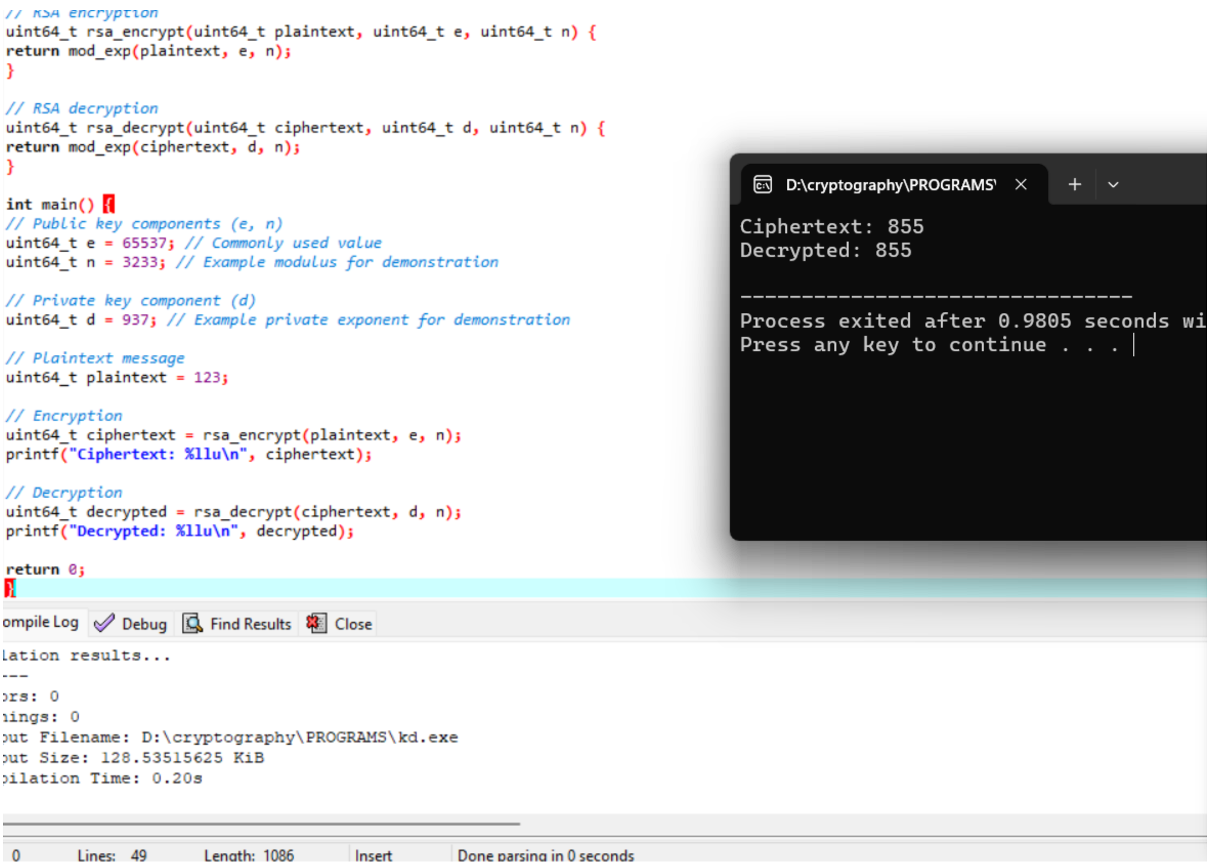
9. Implement DES encryption in CFB mode using the provided plaintext, key, and IV.
10. This algorithm outlines the steps performed by the code to encrypt a plaintext message

Program :

```
#include <stdio.h>
#include <stdint.h>

// Modular exponentiation
uint64_t mod_exp(uint64_t base, uint64_t exp, uint64_t mod) {
uint64_t result = 1;
while (exp > 0) {
if (exp % 2 == 1)
result = (result * base) % mod;
base = (base * base) % mod;
exp /= 2;
}
return result;
}

// RSA encryption
uint64_t rsa_encrypt(uint64_t plaintext, uint64_t e, uint64_t n) {
return mod_exp(plaintext, e, n);
}

// RSA decryption
uint64_t rsa_decrypt(uint64_t ciphertext, uint64_t d, uint64_t n) {
return mod_exp(ciphertext, d, n);
}

int main() {
// Public key components (e, n)
uint64_t e = 65537; // Commonly used value
uint64_t n = 3233; // Example modulus for demonstration
```

```
// Private key component (d)
uint64_t d = 937; // Example private exponent for demonstration

// Plaintext message
uint64_t plaintext = 123;

// Encryption
uint64_t ciphertext = rsa_encrypt(plaintext, e, n);
printf("Ciphertext: %llu\n", ciphertext);

// Decryption
uint64_t decrypted = rsa_decrypt(ciphertext, d, n);
printf("Decrypted: %llu\n", decrypted);

return 0;
```

Input and Output:



Result : implementation of a c program RSA Implement a program that generates RSA public and privatekeys, and performs encryption and decryption of messages is sucessfull