CRYPTOGRAPHY AND NETWORK SECURITY
LAB - 2

Name of the Student: SreeDananjay S
Registration Number: 21BAI1807
Slot: L31+L32
Course Code: BCSE309P
Programme: Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning
School: School of Computer Science and Engineering(SCOPE)

Q2)

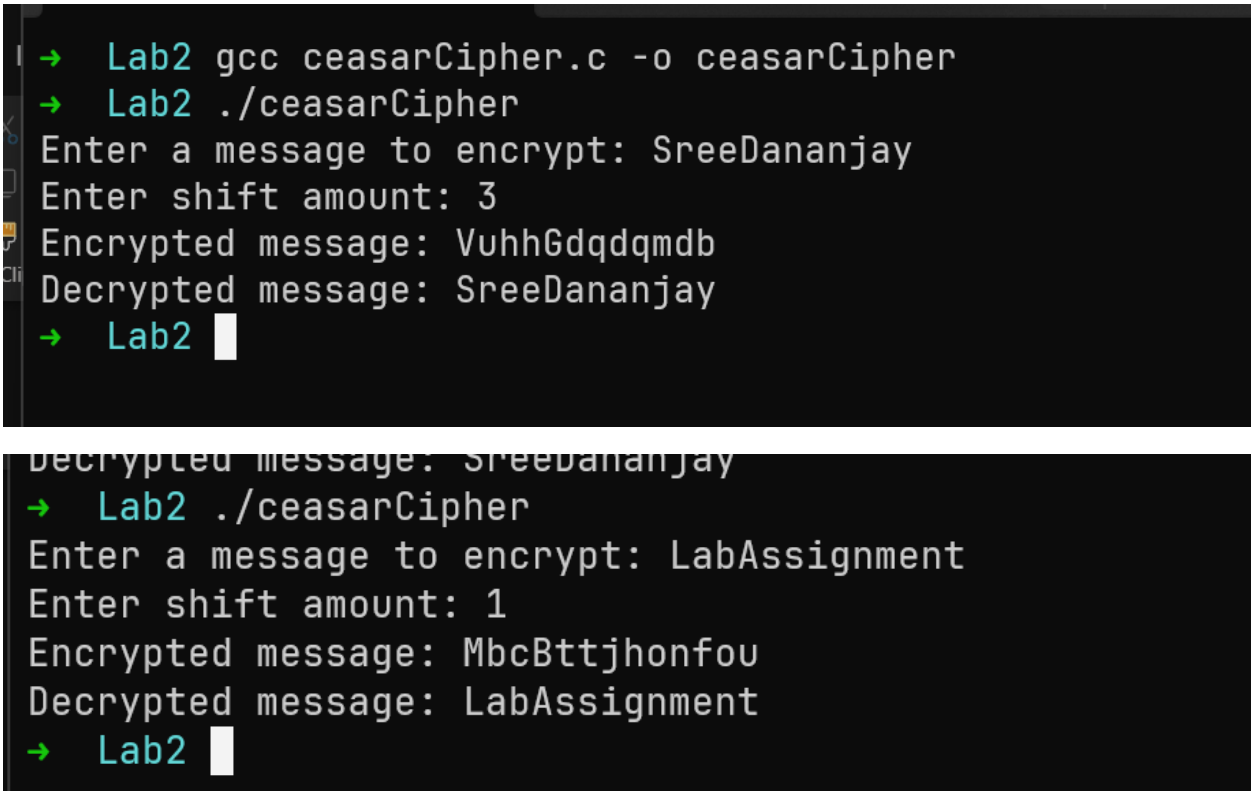
a) Caesar Cipher algorithm implementation in C

Code:

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

void encrypt(char *text, int shift) {
  char ch;
  for (int i = 0; text[i] != '\0'; ++i) {
    ch = text[i];

    // Encrypt uppercase letters
    if (isupper(ch)) {
      text[i] = (ch + shift - 'A') % 26 + 'A';
    }
    // Encrypt lowercase letters
    else if (islower(ch)) {
      text[i] = (ch + shift - 'a') % 26 + 'a';
    }
  }
}

void decrypt(char *text, int shift) {
  encrypt(text, 26 - shift);
}

int main() {
  char text[100];
  int shift;

  printf("Enter a message to encrypt: ");
  fgets(text, sizeof(text), stdin);
  text[strcspn(text, "\n")] = 0; // Remove newline character
```

```
    printf("Enter shift amount: ");
    scanf("%d", &shift);

    encrypt(text, shift);
    printf("Encrypted message: %s\n", text);

    decrypt(text, shift);
    printf("Decrypted message: %s\n", text);

    return 0;
}
```

Sample output Screenshot:



b) Playfair Cipher Implementation in C

Code:

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

```
#include <string.h>
#include <ctype.h>

#define SIZE 5

void generateKeyTable(const char* key, char keyTable[SIZE][SIZE]) {
  int letterExists[26] = {0}; // Track letters already added to the table
  int index = 0;
  int row = 0, col = 0;

  // Add key letters to the table
  for (int i = 0; key[i] != '\0'; i++) {
    char ch = toupper(key[i]);

    if (ch == 'J') ch = 'I'; // Treat 'J' as 'I'

    if (!letterExists[ch - 'A']) {
      keyTable[row][col] = ch;
      letterExists[ch - 'A'] = 1;

      col++;
      if (col == SIZE) {
        col = 0;
        row++;
      }
    }
  }

  // Add remaining letters to the table
  for (char ch = 'A'; ch <= 'Z'; ch++) {
    if (ch == 'J') continue; // Skip 'J'

    if (!letterExists[ch - 'A']) {
      keyTable[row][col] = ch;
      letterExists[ch - 'A'] = 1;

      col++;
      if (col == SIZE) {
```

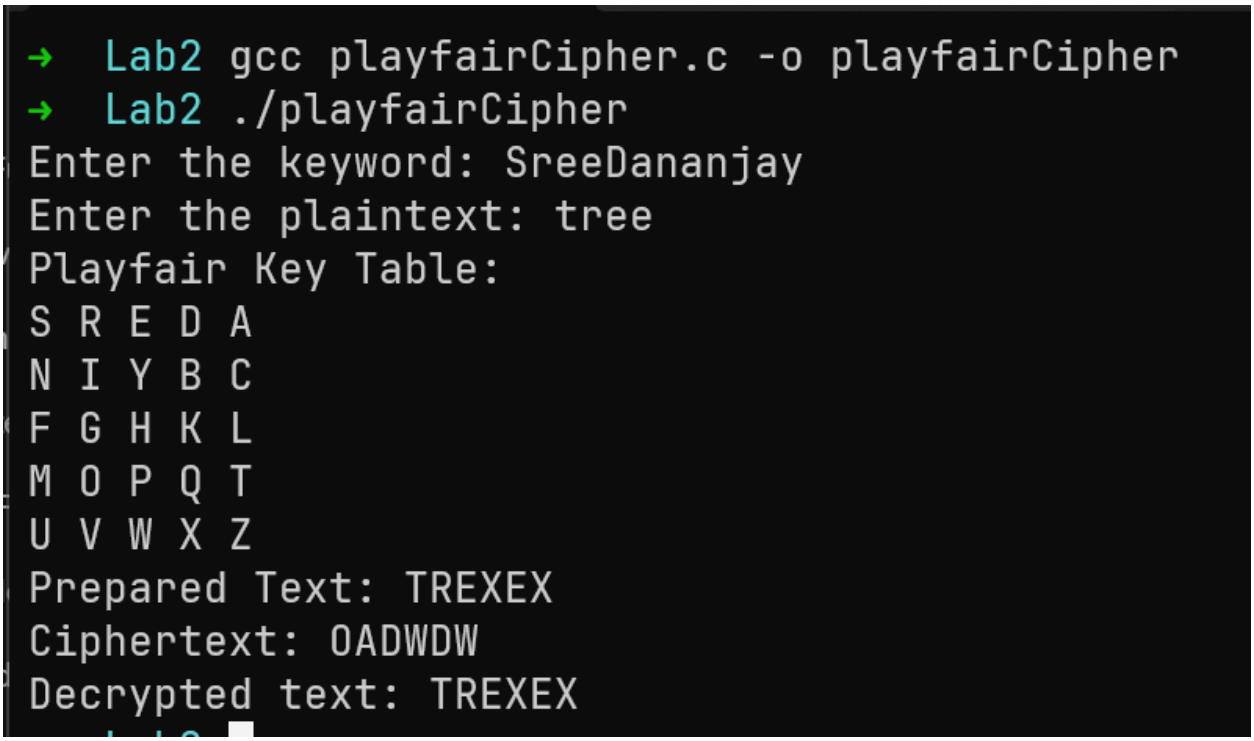
```
            col = 0;
            row++;
        }
    }
  }
}

void printKeyTable(char keyTable[SIZE][SIZE]) {
  printf("Playfair Key Table:\n");
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      printf("%c ", keyTable[i][j]);
    }
    printf("\n");
  }
}

void findPosition(char keyTable[SIZE][SIZE], char ch, int* row, int* col) {
  if (ch == 'J') ch = 'I'; // Treat 'J' as 'I'

  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      if (keyTable[i][j] == ch) {
        *row = i;
        *col = j;
        return;
      }
    }
  }
}

void prepareText(const char* input, char* prepared) {
  int len = strlen(input);
  int index = 0;

  for (int i = 0; i < len; i++) {
    char ch = toupper(input[i]);
```

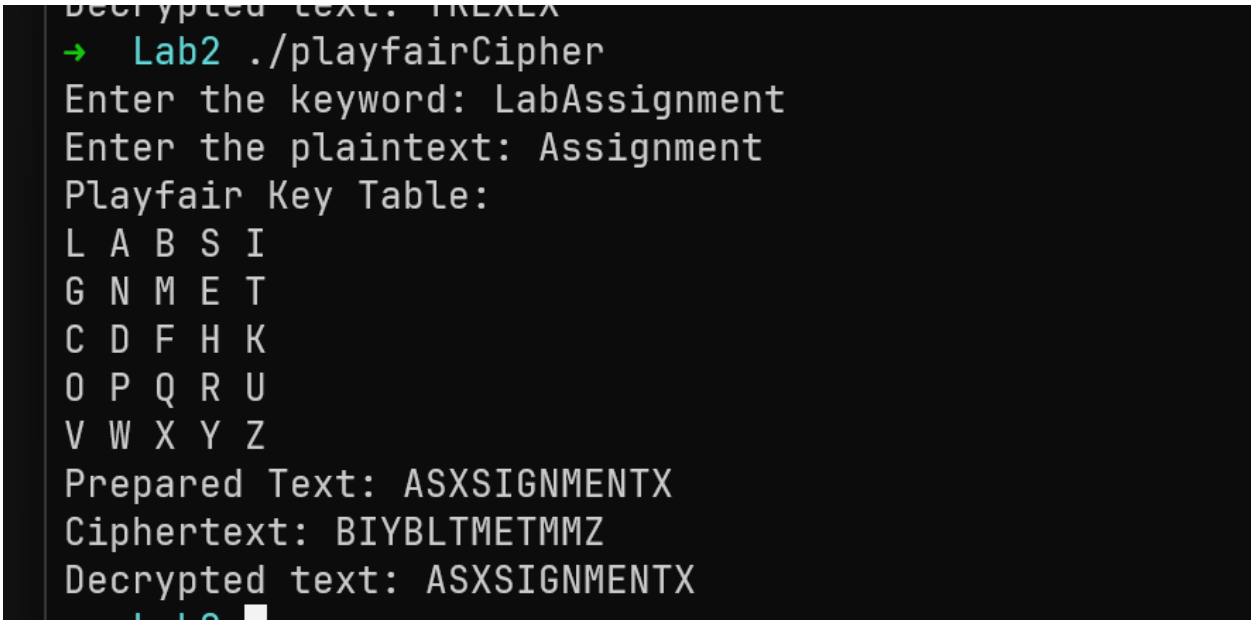
```
        if (ch == 'J') ch = 'I'; // Treat 'J' as 'I'

        if (isalpha(ch)) {
            if (index > 0 && prepared[index - 1] == ch) {
                prepared[index++] = 'X'; // Insert 'X' between duplicate letters
            }
            prepared[index++] = ch;
        }
    }

    if (index % 2 != 0) {
        prepared[index++] = 'X'; // Add 'X' if length is odd
    }

    prepared[index] = '\0';
}

void encryptDigraph(const char digraph[2], char* result, char keyTable[SIZE][SIZE]) {
    int row1, col1, row2, col2;
    findPosition(keyTable, digraph[0], &row1, &col1);
    findPosition(keyTable, digraph[1], &row2, &col2);

    if (row1 == row2) {
        // Same row, shift right
        result[0] = keyTable[row1][(col1 + 1) % SIZE];
        result[1] = keyTable[row2][(col2 + 1) % SIZE];
    } else if (col1 == col2) {
        // Same column, shift down
        result[0] = keyTable[(row1 + 1) % SIZE][col1];
        result[1] = keyTable[(row2 + 1) % SIZE][col2];
    } else {
        // Rectangle case
        result[0] = keyTable[row1][col2];
        result[1] = keyTable[row2][col1];
    }
}

void decryptDigraph(const char digraph[2], char* result, char keyTable[SIZE][SIZE]) {
```

```
    int row1, col1, row2, col2;
    findPosition(keyTable, digraph[0], &row1, &col1);
    findPosition(keyTable, digraph[1], &row2, &col2);

    if (row1 == row2) {
        // Same row, shift left
        result[0] = keyTable[row1][(col1 + SIZE - 1) % SIZE];
        result[1] = keyTable[row2][(col2 + SIZE - 1) % SIZE];
    } else if (col1 == col2) {
        // Same column, shift up
        result[0] = keyTable[(row1 + SIZE - 1) % SIZE][col1];
        result[1] = keyTable[(row2 + SIZE - 1) % SIZE][col2];
    } else {
        // Rectangle case
        result[0] = keyTable[row1][col2];
        result[1] = keyTable[row2][col1];
    }
}

void playfairEncrypt(const char* plaintext, char* ciphertext, char
keyTable[SIZE][SIZE]) {
    char prepared[200];
    prepareText(plaintext, prepared);
    printf("Prepared Text: %s\n", prepared);

    int len = strlen(prepared);
    for (int i = 0; i < len; i += 2) {
        encryptDigraph(&prepared[i], &ciphertext[i], keyTable);
    }
    ciphertext[len] = '\0';
}

void playfairDecrypt(const char* ciphertext, char* plaintext, char
keyTable[SIZE][SIZE]) {
    int len = strlen(ciphertext);
    for (int i = 0; i < len; i += 2) {
        decryptDigraph(&ciphertext[i], &plaintext[i], keyTable);
    }
```

```
    plaintext[len] = '\0';
}

int main() {
    char key[100], plaintext[200], ciphertext[200], decrypted[200];
    char keyTable[SIZE][SIZE];

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

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

    generateKeyTable(key, keyTable);
    printKeyTable(keyTable);

    playfairEncrypt(plaintext, ciphertext, keyTable);
    printf("Ciphertext: %s\n", ciphertext);

    playfairDecrypt(ciphertext, decrypted, keyTable);
    printf("Decrypted text: %s\n", decrypted);

    return 0;
}
```

Output Screenshots:





Result:

Thus, both encryption algorithms have been executed and verified successfully.