Lab Report

of

CRYPTOGRAPHY

Subject Code: CSC 327

Submitted To

SOCH COLLEGE OF IT

(AFFILIATED TO TRIBHUVAN UNIVERSITY)

Ranipauwa, Pokhara – 11

Submitted By

Amrit Karki

Registration No: 79011964

Program: Bachelor of Science in Computer Science and Information Technology (BSc. CSIT)

Semester: Fifth

List of Exercises

S/ N	Title of Experiment	Date	Remarks	Page Number
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Faculty Name:

Faculty Signature:

Lab Administrator Name:

Lab Administrator Signature:

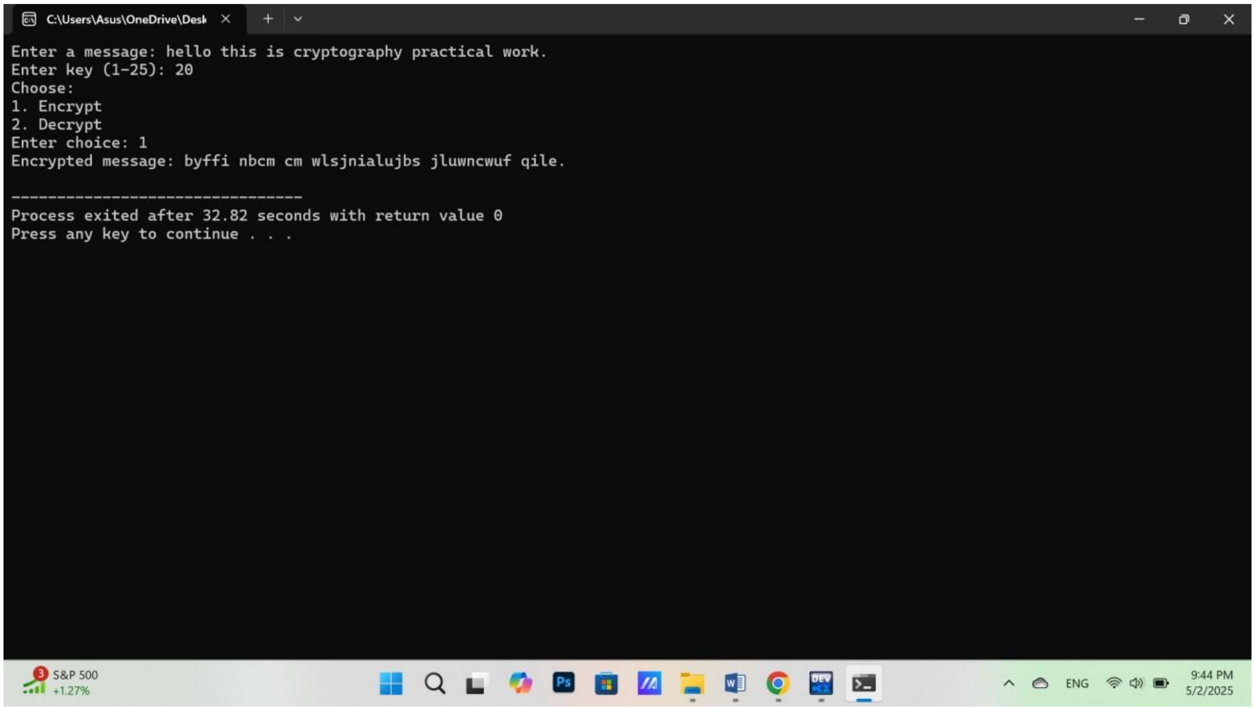
External Examiner Name:

External Examiner Signature:

Executable Code:

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

void encrypt(char *message, int key) {     for (int i = 0;
message[i] != '\0'; i++) {         if (isalpha(message[i])) {
char base = isupper(message[i]) ? 'A' : 'a';
message[i] = (message[i] - base + key) % 26 + base;
        }
    }
}

void decrypt(char *message, int key) {     for (int i = 0;
message[i] != '\0'; i++) {         if (isalpha(message[i])) {
char base = isupper(message[i]) ? 'A' : 'a';             message[i] =
(message[i] - base - key + 26) % 26 + base;
        }
    }
}

int main() {     char
message[100];     int
key, choice;
printf("Enter a
message: ");
fgets(message,
sizeof(message),
stdin);
message[strcspn(me
ssage, "\n")] = '\0';
// remove newline

    printf("Enter key (1-25): ");
scanf("%d", &key);
    key = key % 26; // Ensure key is within 0-25

    printf("Choose:\n1. Encrypt\n2. Decrypt\nEnter choice: ");
scanf("%d", &choice);

    if (choice == 1) {        encrypt(message, key);
printf("Encrypted message: %s\n", message);
    } else if (choice == 2) {        decrypt(message,
key);        printf("Decrypted message: %s\n",
message);
    } else {
        printf("Invalid choice.\n");
    }

    return 0;
}
```

Output:



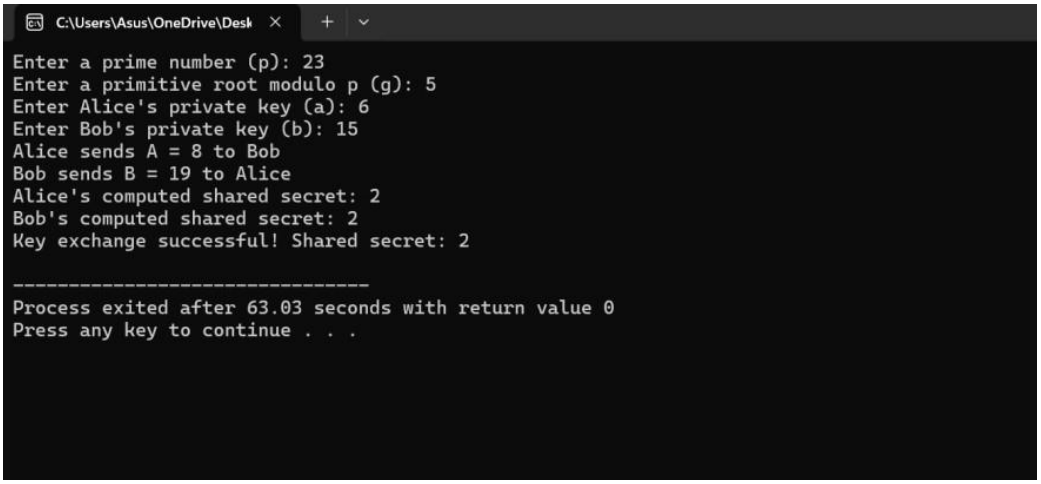
```
Enter a message: hello this is cryptography practical work.
Enter key (1-25): 20
Choose:
1. Encrypt
2. Decrypt
Enter choice: 1
Encrypted message: byffi nbcm cm wlsjnialujbs jluwncwuf qile.

--------------------------------
Process exited after 32.82 seconds with return value 0
Press any key to continue . . .
```

Executable code:

```
#include <stdio.h>

#include <math.h>

// Function to perform modular exponentiation (base^exp % mod)

long long power(long long base, long long exp, long long mod) {
long long result = 1;     base = base % mod;     while (exp > 0) {         if
(exp % 2 == 1) // if exp is odd            result = (result * base) % mod;
exp = exp >> 1; // exp = exp / 2         base = (base * base) % mod;

    }     return
result;

}



int main() {

    long long p, g, a, b, A, B, secretA, secretB;



    // Publicly known values     printf("Enter
a prime number (p): ");     scanf("%lld",
&p);



    printf("Enter a primitive root modulo p (g): ");
scanf("%lld", &g);


    // Alice's private key

    printf("Enter Alice's private key (a): ");
scanf("%lld", &a);



    // Bob's private key

    printf("Enter Bob's private key (b): ");
scanf("%lld", &b);
```

```
    // Alice computes A = g^a mod p     A
= power(g, a, p);

    printf("Alice sends A = %lld to Bob\n", A);


    // Bob computes B = g^b mod p     B
= power(g, b, p);

    printf("Bob sends B = %lld to Alice\n", B);


    // Each computes the shared secret
secretA = power(B, a, p); // (B^a) mod p
secretB = power(A, b, p); // (A^b) mod p


    printf("Alice's computed shared secret: %lld\n", secretA);     printf("Bob's
computed shared secret: %lld\n", secretB);


    if (secretA == secretB)

        printf("Key exchange successful! Shared secret: %lld\n", secretA);     else

        printf("Key exchange failed!\n");



    return 0;

}
```

Output:

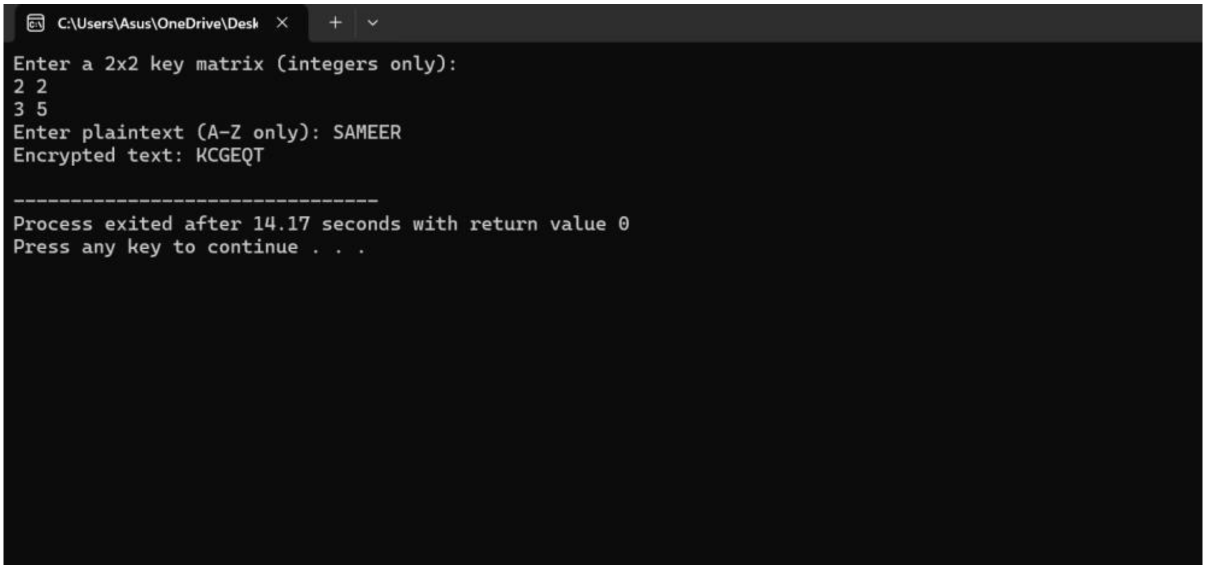


```
Enter a prime number (p): 23
Enter a primitive root modulo p (g): 5
Enter Alice's private key (a): 6
Enter Bob's private key (b): 15
Alice sends A = 8 to Bob
Bob sends B = 19 to Alice
Alice's computed shared secret: 2
Bob's computed shared secret: 2
Key exchange successful! Shared secret: 2

--------------------------------
Process exited after 63.03 seconds with return value 0
Press any key to continue . . .
```

Executable code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>


#define SIZE 2


// Function to multiply key matrix and plaintext vector void
encrypt(char plaintext[], int key[SIZE][SIZE]) {     int i, j, k;

    int len = strlen(plaintext);


    // Make sure length is even (pad with 'X' if odd)
if (len % 2 != 0) {        plaintext[len] = 'X';
plaintext[len + 1] = '\0';        len++;

    }


    printf("Encrypted text: ");     for
(i = 0; i < len; i += 2) {

        int p[2] = { toupper(plaintext[i]) - 'A', toupper(plaintext[i+1]) - 'A' };        int
c[2] = {0};


        for (j = 0; j < SIZE; j++) {
for (k = 0; k < SIZE; k++) {                c[j]
+= key[j][k] * p[k];

            }           c[j]
%= 26;

        }


        printf("%c%c", c[0] + 'A', c[1] + 'A');
    }
printf("\n");

}


int main() {     char
plaintext[100];     int
key[SIZE][SIZE];


    printf("Enter a 2x2 key matrix (integers only):\n");
for (int i = 0; i < SIZE; i++)         for (int j = 0; j < SIZE; j++)
scanf("%d", &key[i][j]);


    printf("Enter plaintext (A-Z only): ");     scanf("%s",
plaintext);


    encrypt(plaintext, key);


    return 0;

}
```

Output:



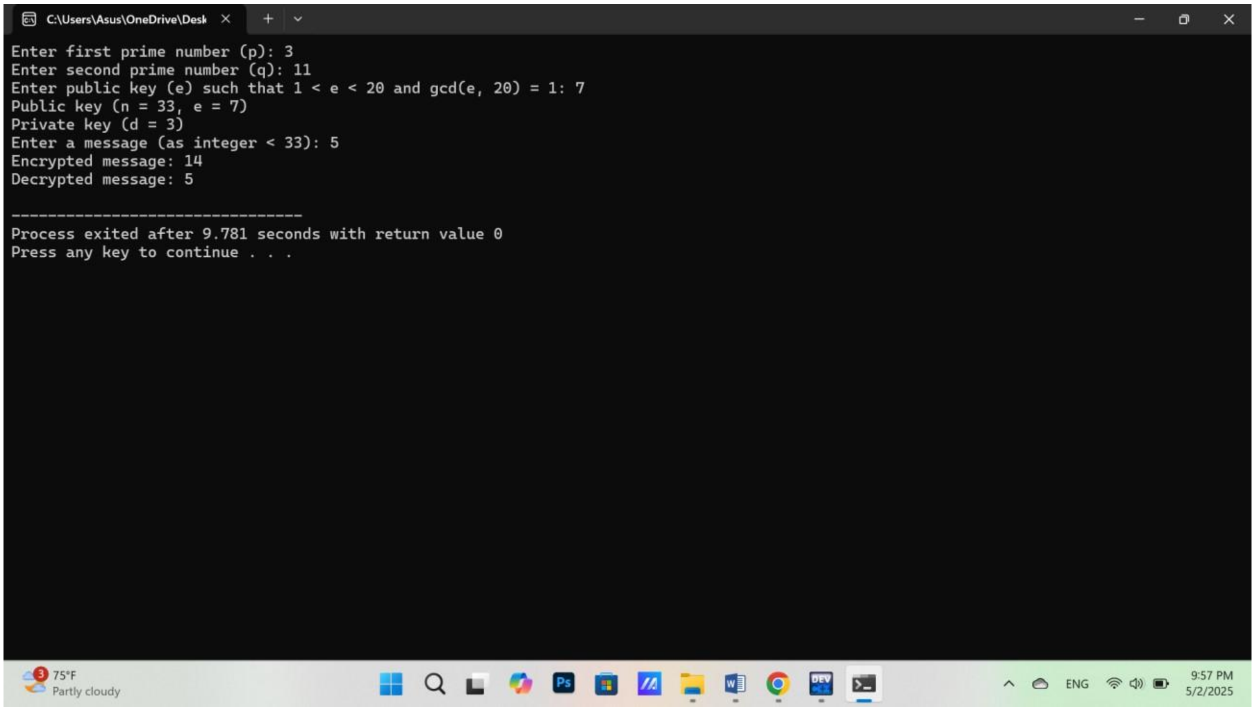
```
Enter a 2x2 key matrix (integers only):
2 2
3 5
Enter plaintext (A-Z only): SAMEER
Encrypted text: KCGEQT

--------------------------------
Process exited after 14.17 seconds with return value 0
Press any key to continue . . .
```

Executable Code:

```
#include <stdio.h>


// Function to calculate gcd
int gcd(int a, int b) {     while
(b != 0) {         int temp = b;
b = a % b;         a = temp;

    }
return a;

}


// Function to find modular inverse of e mod phi (brute-force) int
modInverse(int e, int phi) {     for (int d = 1; d < phi; d++) {         if
((e * d) % phi == 1)             return d;

    }
return -1;

}


// Function to perform modular exponentiation (base^exp % mod) long
long modExp(long long base, long long exp, long long mod) {     long
long result = 1;     base = base % mod;     while (exp > 0) {

        if (exp % 2 == 1)

            result = (result * base) % mod;
exp = exp >> 1;         base = (base * base)
% mod;

    }     return
result;

}


int main() {     int p, q,
n, phi, e, d;     int
message;
```

```
    long long encrypted, decrypted;


    // Example small prime numbers     printf("Enter
first prime number (p): ");     scanf("%d", &p);


    printf("Enter second prime number (q): ");
scanf("%d", &q);


    n = p * q;     phi = (p -
1) * (q - 1);


    // Choose public key e

    printf("Enter public key (e) such that 1 < e < %d and gcd(e, %d) = 1: ", phi, phi);     scanf("%d", &e);


    if (gcd(e, phi) != 1) {

        printf("Invalid e. It must be coprime with %d\n", phi);         return
1;

    }


    // Calculate private key d
d = modInverse(e, phi);     if
(d == -1) {

        printf("Modular inverse for e doesn't exist.\n");
return 1;

    }


    printf("Public key (n = %d, e = %d)\n", n, e);
printf("Private key (d = %d)\n", d);


    // Message input
    printf("Enter a message (as integer < %d): ", n);
scanf("%d", &message);


    // Encryption: c = m^e mod n     encrypted =
modExp(message, e, n);     printf("Encrypted message:
%lld\n", encrypted);


    // Decryption: m = c^d mod n     decrypted =
modExp(encrypted, d, n);     printf("Decrypted
message: %lld\n", decrypted);     return 0;

}
```

Output:



```
Enter first prime number (p): 3
Enter second prime number (q): 11
Enter public key (e) such that 1 < e < 20 and gcd(e, 20) = 1: 7
Public key (n = 33, e = 7)
Private key (d = 3)
Enter a message (as integer < 33): 5
Encrypted message: 14
Decrypted message: 5

--------------------------------
Process exited after 9.781 seconds with return value 0
Press any key to continue . . .
```

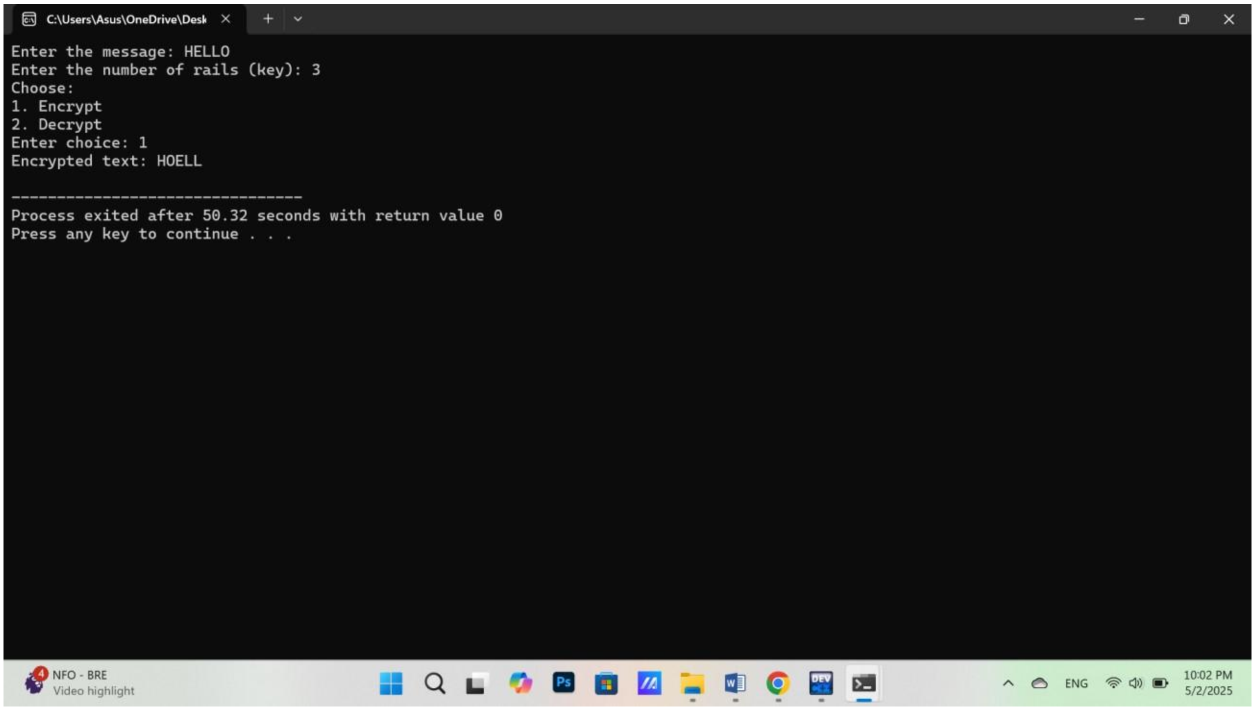
Executable Code:

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>


// Encryption function void
encryptRailFence(char *text, int key) {     int
len = strlen(text);     char rail[key][len];


    // Filling rail matrix with '\n'
for (int i = 0; i < key; i++)         for
(int j = 0; j < len; j++)
rail[i][j] = '\n';


    // To determine the direction
int row = 0, dir_down = 0;     for
(int i = 0; i < len; i++) {
//Place

rail[row][i] = text[i];


// Change direction if top or bottom

if (row == 0 || row == key – 1

dir_down = !dir_down;


// Move up or

row += dir_down ? 1 : -1;     }

  // Read the rail matrix row-wise to get ciphertext

 printf("Encrypted text: ");     for (int i = 0; i < key; i++)

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

if (rail[i][j] != '\n')
```

```
 printf("%c", rail[i][j]);     printf("\n"); }


// Decryption function void
decryptRailFence(char *cipher, int key) {     int len
= strlen(cipher);     char rail[key][len];

   // Fill with '\n'     for (int i = 0;
i < key; i++)         for (int j = 0; j <
len; j++)            rail[i][j] = '\n';

   // Mark the path with '*'
int row = 0, dir_down = 0;     for
(int i = 0; i < len; i++) {
rail[row][i] = '*';


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

           dir_down = !dir_down;


       row += dir_down ? 1 : -1;

   }

   // Fill the '*' positions with actual ciphertext
int idx = 0;     for (int i = 0; i < key; i++)         for
(int j = 0; j < len; j++)            if (rail[i][j] == '*')
rail[i][j] = cipher[idx++];


   // Read the matrix in zigzag to reconstruct original message
printf("Decrypted text: ");     row = 0;     dir_down = 0;     for (int i
= 0; i < len; i++) {         printf("%c", rail[row][i]);


       if (row == 0 || row == key - 1)
dir_down = !dir_down;


       row += dir_down ? 1 : -1;
```

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

}


int main() {     char
message[100];     int
choice, key;


    printf("Enter the message: ");     scanf("%s",
message);


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


    printf("Choose:\n1. Encrypt\n2. Decrypt\nEnter choice: ");     scanf("%d",
&choice);


    if (choice == 1)
encryptRailFence(message, key);     else if
(choice == 2)         decryptRailFence(message,
key);     else

        printf("Invalid choice.\n");


    return 0;

}
```

Output:



```
Enter the message: HELLO
Enter the number of rails (key): 3
Choose:
1. Encrypt
2. Decrypt
Enter choice: 1
Encrypted text: HOELL

--------------------------------
Process exited after 50.32 seconds with return value 0
Press any key to continue . . .
```