FACULTY OF ENGINEERING AND TECHNOLOGY
BACHELOR OF TECHNOLOGY

Information and Network Security Laboratory (303105376)

SEMESTER VII
Computer Science and Engineering Department

Lab Manual

CERTIFICATE

*Mr. **Abhishek Kumar Singh** with Enrollment No. **2203051050947** has successfully completed his laboratory experiments **Information and Network Security Laboratory (303105376)** from the department of **Computer Science and Engineering** during the academic year **2025-2026.***

Date of Submission ………………. **Staff In charge ………………**

Head of Department………………

INDEX

Practical: 1

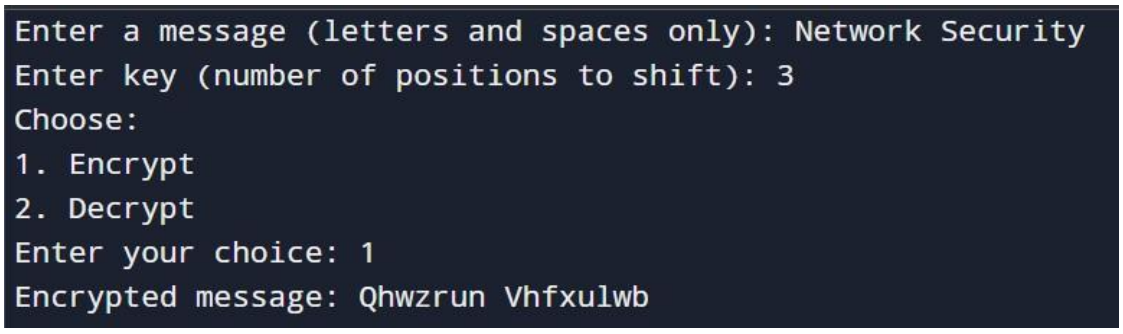
Implement Caesar cipher encryption-decryption

Code:

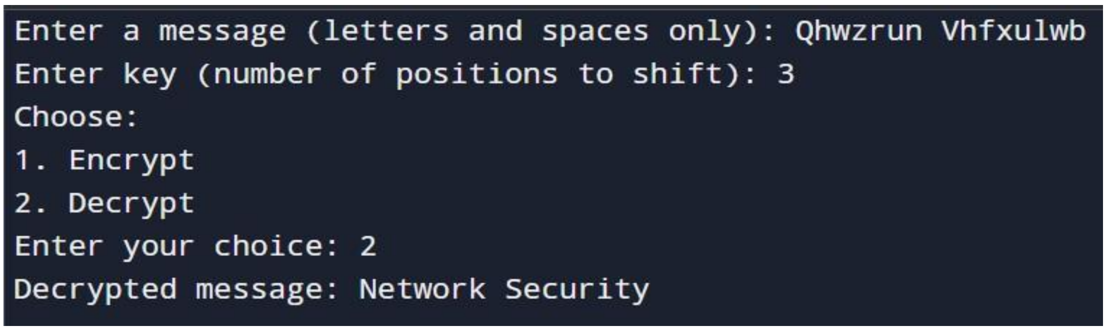
```
#include <stdio.h>
int main() {
    char message[100];
    int key, i, choice;
    printf("Enter a message (letters and spaces only): ");
    scanf(" %[^\n]", message); // Reads line including spaces
    printf("Enter key (number of positions to shift): ");
    scanf("%d", &key);
    printf("Choose:\n1. Encrypt\n2. Decrypt\nEnter your choice: ");
    scanf("%d", &choice);
    for (i = 0; message[i] != '\0'; i++) {
        char ch = message[i];
        if (ch >= 'A' && ch <= 'Z') {
            if (choice == 1)
                message[i] = ((ch - 'A' + key) % 26) + 'A';
            else if (choice == 2)
                message[i] = ((ch - 'A' - key + 26) % 26) + 'A';
        }
        else if (ch >= 'a' && ch <= 'z') {
            if (choice == 1)
                message[i] = ((ch - 'a' + key) % 26) + 'a';
            else if (choice == 2)
                message[i] = ((ch - 'a' - key + 26) % 26) + 'a';
        }
        // Spaces and other characters remain unchanged
    }
    if (choice == 1)
```

```
        printf("Encrypted message: %s\n", message);
    else if (choice == 2)
        printf("Decrypted message: %s\n", message);
    else
        printf("Invalid choice!\n");
    return 0;
}
```

Output:



```
Enter a message (letters and spaces only): Network Security
Enter key (number of positions to shift): 3
Choose:
1. Encrypt
2. Decrypt
Enter your choice: 1
Encrypted message: Qhwzrun Vhfxulwb
```



```
Enter a message (letters and spaces only): Qhwzrun Vhfxulwb
Enter key (number of positions to shift): 3
Choose:
1. Encrypt
2. Decrypt
Enter your choice: 2
Decrypted message: Network Security
```

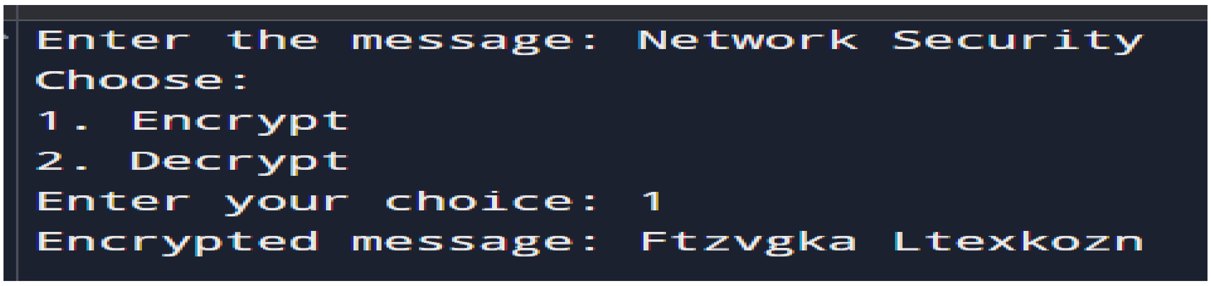
Practical: 2

Implement Monoalphabetic cipher encryption-decryption

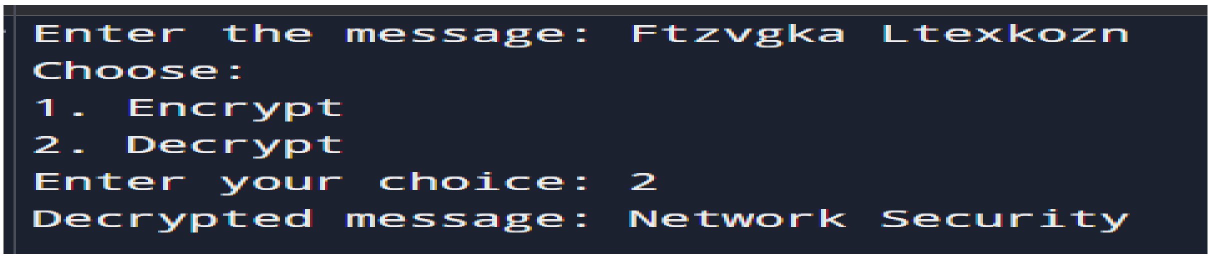
Code:

```
#include <stdio.h>
#include <string.h>
void encrypt(char message[], char key[]) {
    for (int i = 0; message[i] != '\0'; i++) {
        char ch = message[i];
        if (ch >= 'A' && ch <= 'Z') {
            // Uppercase encryption
            message[i] = key[ch - 'A'];
        } else if (ch >= 'a' && ch <= 'z') {
            // Lowercase encryption (use same key and convert to lowercase)
            message[i] = key[ch - 'a'] + 32;
        }
        // Other characters (spaces, punctuation) remain unchanged
    }
}
void decrypt(char message[], char key[]) {
    char reverseKey[26];
    // // Create reverse mapping for uppercase
    for (int i = 0; i < 26; i++) {
        reverseKey[key[i] - 'A'] = 'A' + i;
    }
    for (int i = 0; message[i] != '\0'; i++) {
        char ch = message[i];
        if (ch >= 'A' && ch <= 'Z') {
            // Uppercase decryption
            message[i] = reverseKey[ch - 'A'];
        } else if (ch >= 'a' && ch <= 'z') {
            // Lowercase decryption
            message[i] = reverseKey[ch - 'a'] + 32;
        }
        // Other characters remain unchanged
    }
}
int main() {
    char message[100];
    // Fixed substitution key (must be a permutation of A-Z)
    char key[26] = {
        'Q','W','E','R','T','Y','U','I','O','P',
        'A','S','D','F','G','H','J','K','L','Z',
        'X','C','V','B','N','M'
    };
    int choice;
    printf("Enter the message: ");
    scanf(" %[^\n]", message); // Read line with spaces
    printf("Choose:\n1. Encrypt\n2. Decrypt\nEnter your 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 the message: Network Security
Choose:
1. Encrypt
2. Decrypt
Enter your choice: 1
Encrypted message: Ftzvgka Ltexkozn
```



```
Enter the message: Ftzvgka Ltexkozn
Choose:
1. Encrypt
2. Decrypt
Enter your choice: 2
Decrypted message: Network Security
```

Practical: 3

Implement Playfair cipher encryption-decryption

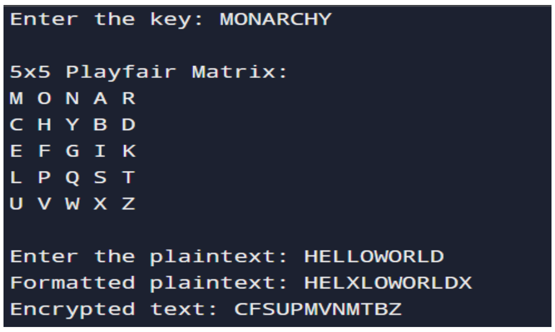
Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
char matrix[5][5];
// Removes duplicate characters from keyword
void prepareKey(char *key, char *result) {
    int used[26] = {0};
    int k = 0;
    for (int i = 0; key[i] != '\0'; i++) {
        char ch = toupper(key[i]);
        if (ch == 'J') ch = 'I';  // Treat I and J as same
         if (ch >= 'A' && ch <= 'Z' && !used[ch - 'A']) {
            result[k++] = ch;
            used[ch - 'A'] = 1;
        }
    }
     // Add remaining letters
    for (char ch = 'A'; ch <= 'Z'; ch++) {
        if (ch == 'J') continue;
        if (!used[ch - 'A']) {
            result[k++] = ch;
            used[ch - 'A'] = 1;
        }
    }
     result[k] = '\0';
}
// Builds 5x5 matrix from cleaned key
```

```
void buildMatrix(char *key) {
    int idx = 0;
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 5; j++)
            matrix[i][j] = key[idx++];
}
// Formats plaintext: removes spaces, handles duplicate pairs
void formatPlaintext(char *input, char *output) {
    int k = 0;
    for (int i = 0; input[i] != '\0'; i++) {
        char ch = toupper(input[i]);
        if (ch < 'A' || ch > 'Z') continue;
        if (ch == 'J') ch = 'I';
        output[k++] = ch;
    }
    output[k] = '\0';
     // Insert 'X' between duplicate letters in a pair
    char temp[100];
    int t = 0;
    for (int i = 0; i < k; i++) {
        temp[t++] = output[i];
        if (i + 1 < k && output[i] == output[i + 1]) {
            temp[t++] = 'X';
        }
    }
    if (t % 2 != 0)
        temp[t++] = 'X';
    temp[t] = '\0';
    strcpy(output, temp);
}
```

```
// Finds position of a character in the matrix
void findPosition(char ch, int *row, int *col) {
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 5; j++)
            if (matrix[i][j] == ch) {
                *row = i;
                *col = j;
                return;
            }
}
// Encrypts using Playfair rules
void encrypt(char *plaintext, char *ciphertext) {
    int len = strlen(plaintext);
    int k = 0;
     for (int i = 0; i < len; i += 2) {
        char a = plaintext[i];
        char b = plaintext[i + 1];
         int r1, c1, r2, c2;
        findPosition(a, &r1, &c1);
        findPosition(b, &r2, &c2);
         if (r1 == r2) {
            ciphertext[k++] = matrix[r1][(c1 + 1) % 5];
            ciphertext[k++] = matrix[r2][(c2 + 1) % 5];
        } else if (c1 == c2) {
            ciphertext[k++] = matrix[(r1 + 1) % 5][c1];
            ciphertext[k++] = matrix[(r2 + 1) % 5][c2];
        } else {
            ciphertext[k++] = matrix[r1][c2];
            ciphertext[k++] = matrix[r2][c1];
        }
    }
    ciphertext[k] = '\0';
}
int main() {
    char key[100], cleanedKey[26], matrixText[100];
    char plaintext[100], formatted[100], ciphertext[100];
     printf("Enter the key: ");
    scanf("%s", key);
     prepareKey(key, cleanedKey);
    buildMatrix(cleanedKey);
     printf("\n5x5 Playfair Matrix:\n");
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++)
printf("%c ", matrix[i][j]);
printf("\n");
}
printf("\nEnter the plaintext: ");
scanf(" %[^\n]", plaintext);
formatPlaintext(plaintext, formatted);
encrypt(formatted, ciphertext);
printf("Formatted plaintext: %s\n", formatted);
printf("Encrypted text: %s\n", ciphertext);
return 0;
}
```

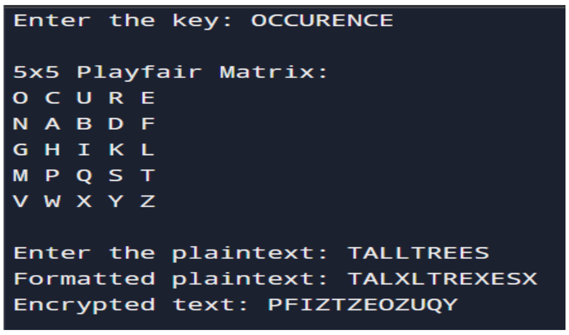
Output:



```
Enter the key: MONARCHY

5x5 Playfair Matrix:
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Enter the plaintext: HELLOWORLD
Formatted plaintext: HELXLOWORLDX
Encrypted text: CFSUPMVNMTBZ
```



```
Enter the key: OCCURENCE

5x5 Playfair Matrix:
O C U R E
N A B D F
G H I K L
M P Q S T
V W X Y Z

Enter the plaintext: TALLTREES
Formatted plaintext: TALXLTREXESX
Encrypted text: PFIZTZEOZUQY
```

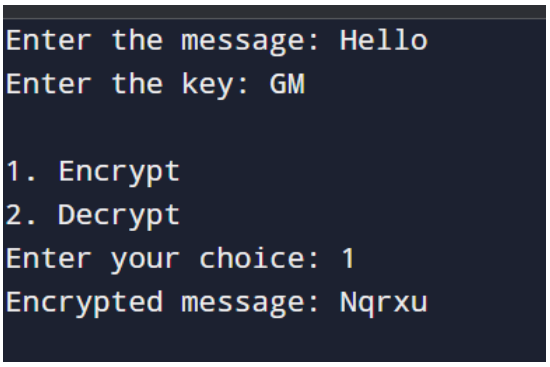
Practical: 4

Implement Polyalphabetic cipher encryption-decryption

Code:

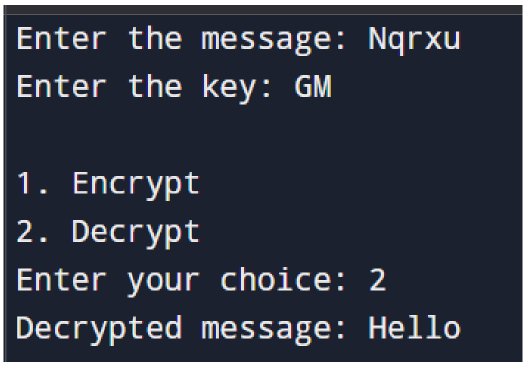
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
// Function to generate repeated key ignoring spaces
void generateKey(char message[], char key[], char newKey[]) {
    int msgLen = strlen(message);
    int keyLen = strlen(key);
    int j = 0;
    for (int i = 0; i < msgLen; i++) {
        if (isalpha(message[i])) {
            newKey[i] = toupper(key[j % keyLen]);
            j++;
        } else {
            newKey[i] = message[i]; // Keep space or punctuation
        }
    }
    newKey[msgLen] = '\0';
}
// Encryption function
void encrypt(char message[], char key[], char result[]) {
    for (int i = 0; message[i] != '\0'; i++) {
        if (isupper(message[i])) {
            result[i] = ((message[i] - 'A') + (key[i] - 'A')) % 26 + 'A';
        } else if (islower(message[i])) {
            result[i] = ((message[i] - 'a') + (key[i] - 'A')) % 26 + 'a';
        } else {
            result[i] = message[i]; // Keep as is
        }
    }
    result[strlen(message)] = '\0';
}
// Decryption function
void decrypt(char cipher[], char key[], char result[]) {
    for (int i = 0; cipher[i] != '\0'; i++) {
        if (isupper(cipher[i])) {
            result[i] = ((cipher[i] - 'A') - (key[i] - 'A') + 26) % 26 + 'A';
        } else if (islower(cipher[i])) {
            result[i] = ((cipher[i] - 'a') - (key[i] - 'A') + 26) % 26 + 'a';
        } else {
            result[i] = cipher[i];
        }
    }
    result[strlen(cipher)] = '\0';
}
int main() {
    char message[100], key[100], newKey[100], result[100];
    int choice;
    printf("Enter the message: ");
    fgets(message, sizeof(message), stdin);
    message[strcspn(message, "\n")] = '\0'; // Remove newline
    printf("Enter the key: ");
    scanf("%s", key);
    printf("\n1. Encrypt\n2. Decrypt\nEnter your choice: ");
    scanf("%d", &choice);
    generateKey(message, key, newKey);
    if (choice == 1) {
        encrypt(message, newKey, result);
        printf("Encrypted message: %s\n", result);
    } else if (choice == 2) {
        decrypt(message, newKey, result);
        printf("Decrypted message: %s\n", result);
    } else {
        printf("Invalid choice!\n");
    }
    return 0;
}
```

Output:



```
Enter the message: Hello
Enter the key: GM

1. Encrypt
2. Decrypt
Enter your choice: 1
Encrypted message: Nqrxu
```



```
Enter the message: Nqrxu
Enter the key: GM

1. Encrypt
2. Decrypt
Enter your choice: 2
Decrypted message: Hello
```

Practical: 5

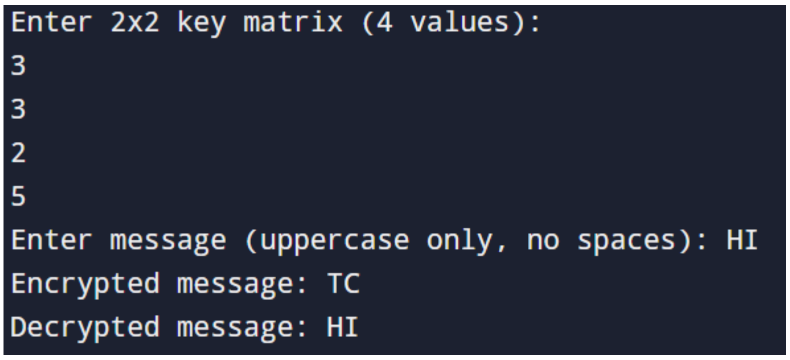
Implement Hill cipher encryption-decryption

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
// Function to find modular inverse of a number modulo 26
int modInverse(int a) {
    a = a % 26;
    for (int x = 1; x < 26; x++) {
        if ((a * x) % 26 == 1)
            return x;
    }
    return -1;
}
// Function to find determinant of 2x2 matrix
int determinant(int key[2][2]) {
    return (key[0][0] * key[1][1] - key[0][1] * key[1][0]) % 26;
}
// Function to calculate inverse of 2x2 matrix mod 26
int inverseKey(int key[2][2], int invKey[2][2]) {
    int det = determinant(key);
    if (det < 0) det += 26; // Adjust negative determinant
    int invDet = modInverse(det);
    if (invDet == -1) {
        printf("Key matrix is not invertible!\n");
        return 0;
    }
    invKey[0][0] = (key[1][1] * invDet) % 26;
    invKey[0][1] = (-key[0][1] * invDet) % 26;
    invKey[1][0] = (-key[1][0] * invDet) % 26;
    invKey[1][1] = (key[0][0] * invDet) % 26;
    // Adjust negative values
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            if (invKey[i][j] < 0)
                invKey[i][j] += 26;
        }
    }
    return 1;
}
// Convert char to number (A=0, B=1, ..., Z=25)
int charToInt(char c) {
    return toupper(c) - 'A';
}
// Convert number to uppercase char
char intToChar(int x) {
    return (x % 26) + 'A';
}
// Encrypt 2-letter block
void encryptBlock(char *block, int key[2][2], char *out) {
    int p[2];
    p[0] = charToInt(block[0]);
    p[1] = charToInt(block[1]);
    int c[2];
    c[0] = (key[0][0] * p[0] + key[0][1] * p[1]) % 26;
    c[1] = (key[1][0] * p[0] + key[1][1] * p[1]) % 26;
    out[0] = intToChar(c[0]);
    out[1] = intToChar(c[1]);
}
```

```
// Decrypt 2-letter block
void decryptBlock(char *block, int invKey[2][2], char *out) {
    int c[2];
    c[0] = charToInt(block[0]);
    c[1] = charToInt(block[1]);
    int p[2];
    p[0] = (invKey[0][0] * c[0] + invKey[0][1] * c[1]) % 26;
    p[1] = (invKey[1][0] * c[0] + invKey[1][1] * c[1]) % 26;
    if (p[0] < 0) p[0] += 26;
    if (p[1] < 0) p[1] += 26;
    out[0] = intToChar(p[0]);
    out[1] = intToChar(p[1]);
}
int main() {
    char message[100], encrypted[100], decrypted[100];
    int key[2][2], invKey[2][2];
    printf("Enter 2x2 key matrix (4 values):\n");
    scanf("%d%d%d%d", &key[0][0], &key[0][1], &key[1][0], &key[1][1]);
    if (!inverseKey(key, invKey))
        return 1; // Stop if key is not invertible
    printf("Enter message (uppercase only, no spaces): ");
    scanf("%s", message);
    int len = strlen(message);
    if (len % 2 != 0) {
        // pad with X
        message[len++] = 'X';
        message[len] = '\0';
    }
    // Encryption
    for (int i = 0; i < len; i += 2)
        encryptBlock(&message[i], key, &encrypted[i]);
    encrypted[len] = '\0';
    printf("Encrypted message: %s\n", encrypted);
    // Decryption
    for (int i = 0; i < len; i += 2)
        decryptBlock(&encrypted[i], invKey, &decrypted[i]);
    decrypted[len] = '\0';
    printf("Decrypted message: %s\n", decrypted);
    return 0;
}
```

Output:



```
Enter 2x2 key matrix (4 values):
3
3
2
5
Enter message (uppercase only, no spaces): HI
Encrypted message: TC
Decrypted message: HI
```

Practical: 6

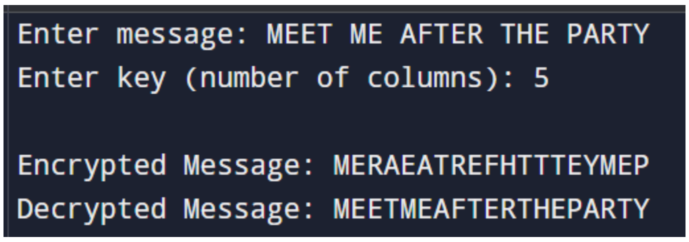
Implement Simple Transposition encryption-decryption

Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
// Encrypts message using simple row-wise transposition
void encrypt(char *message, int key, char *encrypted) {
   int len = strlen(message);
   int rows = (len + key - 1) / key; // ceiling(len/key)
   int k = 0;
   // Read column-wise after filling row-wise
   for (int col = 0; col < key; col++) {
      for (int row = 0; row < rows; row++) {
         int idx = row * key + col;
         if (idx < len) {
            encrypted[k++] = message[idx];
         }
      }
   }
   encrypted[k] = '\0';
}
// Decrypts message using simple row-wise transposition
void decrypt(char *cipher, int key, char *decrypted) {
   int len = strlen(cipher);
   int rows = (len + key - 1) / key;
   int extra = len % key;
   int k = 0;
```

```
    // Matrix to store characters
    char matrix[rows][key];
    memset(matrix, 0, sizeof(matrix));  // Initialize to nulls
    // Fill the matrix column-wise
    for (int col = 0; col < key; col++) {
        int row_limit = (extra != 0 && col >= extra) ? rows - 1 : rows;
        for (int row = 0; row < row_limit; row++) {
            matrix[row][col] = cipher[k++];
        }
    }
    // Read matrix row-wise to get original message
    k = 0;
    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < key; col++) {
            if (matrix[row][col] != 0) {
                decrypted[k++] = matrix[row][col];
            }
        }
    }
    decrypted[k] = '\0';
}
int main() {
    char message[MAX], encrypted[MAX], decrypted[MAX];
    int key;
    printf("Enter message: ");
    fgets(message, MAX, stdin);
    message[strcspn(message, "\n")] = '\0'; // Remove trailing newline
    // Remove spaces and convert to uppercase
    int k = 0;
    for (int i = 0; message[i]; i++) {
        if (message[i] != ' ') {
            message[k++] = toupper(message[i]);
        }
    }
    message[k] = '\0';
    printf("Enter key (number of columns): ");
    scanf("%d", &key);
    encrypt(message, key, encrypted);
    printf("\nEncrypted Message: %s\n", encrypted);
    decrypt(encrypted, key, decrypted);
    printf("Decrypted Message: %s\n", decrypted);
    return 0;
}
```

Output:



```
Enter message: MEET ME AFTER THE PARTY
Enter key (number of columns): 5

Encrypted Message: MERAEATREFHTTTEYMEP
Decrypted Message: MEETMEAFTERTHEPARTY
```

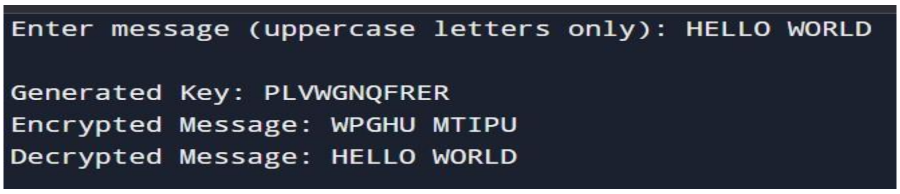
Practical: 7

Implement One time pad encryption-decryption

Code:

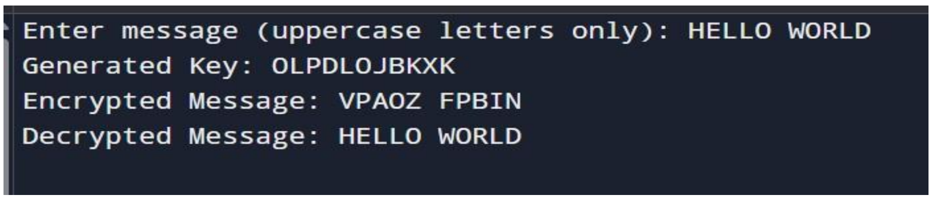
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAX 100
void generateKey(int length, char *key) {
    for (int i = 0; i < length; i++) {
        key[i] = 'A' + (rand() % 26);
    }
    key[length] = '\0';
}
void encrypt(char *message, char *key, char *cipher) {
    int len = strlen(message);
    for (int i = 0; i < len; i++) {
        if (message[i] >= 'A' && message[i] <= 'Z') {
            cipher[i] = ((message[i] - 'A') + (key[i] - 'A')) % 26 + 'A';
        } else {
            cipher[i] = message[i];
        }
    }
    cipher[len] = '\0';
}
void decrypt(char *cipher, char *key, char *message) {
    int len = strlen(cipher);
    for (int i = 0; i < len; i++) {
        if (cipher[i] >= 'A' && cipher[i] <= 'Z') {
            message[i] = ((cipher[i] - 'A') - (key[i] - 'A') + 26) % 26 + 'A';
        } else {
            message[i] = cipher[i];
        }
    }
    message[len] = '\0';
}
int main() {
    char message[MAX], key[MAX], cipher[MAX], decrypted[MAX];
    printf("Enter message (uppercase letters only): ");
    fgets(message, MAX, stdin);
    message[strcspn(message, "\n")] = '\0';
    int len = strlen(message);
    srand(time(0));
    generateKey(len, key);
    printf("Generated Key: %s\n", key);
    encrypt(message, key, cipher);
    printf("Encrypted Message: %s\n", cipher);
    decrypt(cipher, key, decrypted);
    printf("Decrypted Message: %s\n", decrypted);
    return 0;
}
```

Output:



```
Enter message (uppercase letters only): HELLO WORLD

Generated Key: PLVWGNQFRER
Encrypted Message: WPGHU MTIPU
Decrypted Message: HELLO WORLD
```



```
Enter message (uppercase letters only): HELLO WORLD
Generated Key: OLPDLOJBKXK
Encrypted Message: VPAOZ FPBIN
Decrypted Message: HELLO WORLD
```

Practical: 8

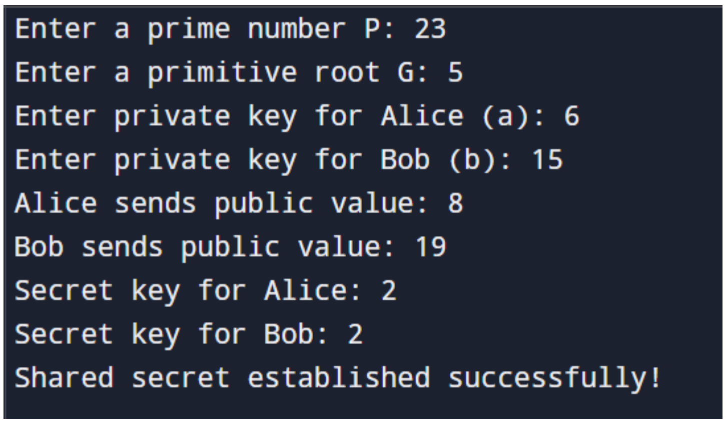
Implement Diffi-Hellmen Key exchange Method

Code:

```
#include <stdio.h>
#include <math.h>
long long power(long long base, long long exp, long long mod) {
    long long result = 1;
    for (int i = 0; i < exp; i++) {
        result = (result * base) % mod;
    }
    return result;
}
int main() {
    long long P, G, a, b;
    long long x, y;
    long long keyA, keyB;
    printf("Enter a prime number P: ");
    scanf("%lld", &P);
    printf("Enter a primitive root G: ");
    scanf("%lld", &G);
    printf("Enter private key for Alice (a): ");
    scanf("%lld", &a);
    printf("Enter private key for Bob (b): ");
    scanf("%lld", &b);

    x = power(G, a, P);
    y = power(G, b, P);
    printf("Alice sends public value: %lld\n", x);
    printf("Bob sends public value: %lld\n", y);
    keyA = power(y, a, P);
    keyB = power(x, b, P);
    printf("Secret key for Alice: %lld\n", keyA);
    printf("Secret key for Bob: %lld\n", keyB);
    if (keyA == keyB) {
        printf("Shared secret established successfully!\n");
    } else {
        printf("Error: Keys do not match.\n");
    }
    return 0;
}
```

Output:



```
Enter a prime number P: 23
Enter a primitive root G: 5
Enter private key for Alice (a): 6
Enter private key for Bob (b): 15
Alice sends public value: 8
Bob sends public value: 19
Secret key for Alice: 2
Secret key for Bob: 2
Shared secret established successfully!
```

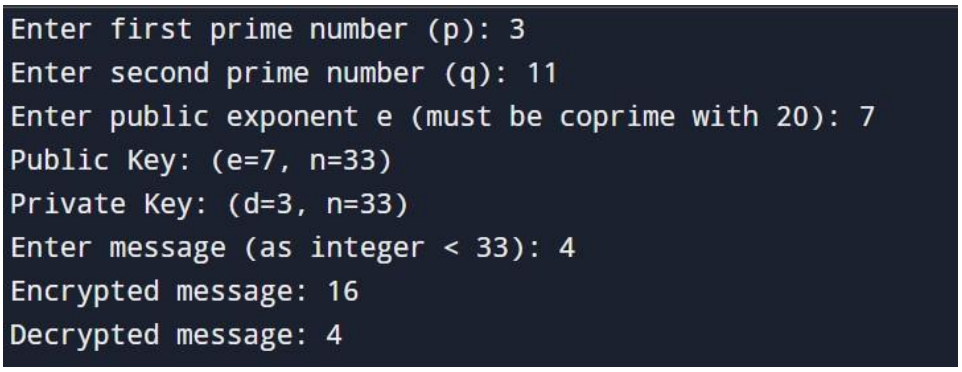
Practical: 9

Implement RSA encryption-decryption algorithm

Code:

```
#include <stdio.h>
#include <math.h>
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
long long power(long long base, long long exp, long long n) {
    long long result = 1;
    for (int i = 0; i < exp; i++) {
        result = (result * base) % n;
    }
    return result;
}
int main() {
    int p, q, n, phi, e, d = 0;
    int msg;
    long long cipher, decrypted;
    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);
    printf("Enter public exponent e (must be coprime with %d): ", phi);
    scanf("%d", &e);
    for (int i = 1; i < phi; i++) {
        if ((e * i) % phi == 1) {
            d = i;
            break;
        }
    }
    printf("Public Key: (e=%d, n=%d)\n", e, n);
    printf("Private Key: (d=%d, n=%d)\n", d, n);
    printf("Enter message (as integer < %d): ", n);
    scanf("%d", &msg);
    cipher = power(msg, e, n);
    printf("Encrypted message: %lld\n", cipher);
    decrypted = power(cipher, 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 exponent e (must be coprime with 20): 7
Public Key: (e=7, n=33)
Private Key: (d=3, n=33)
Enter message (as integer < 33): 4
Encrypted message: 16
Decrypted message: 4
```

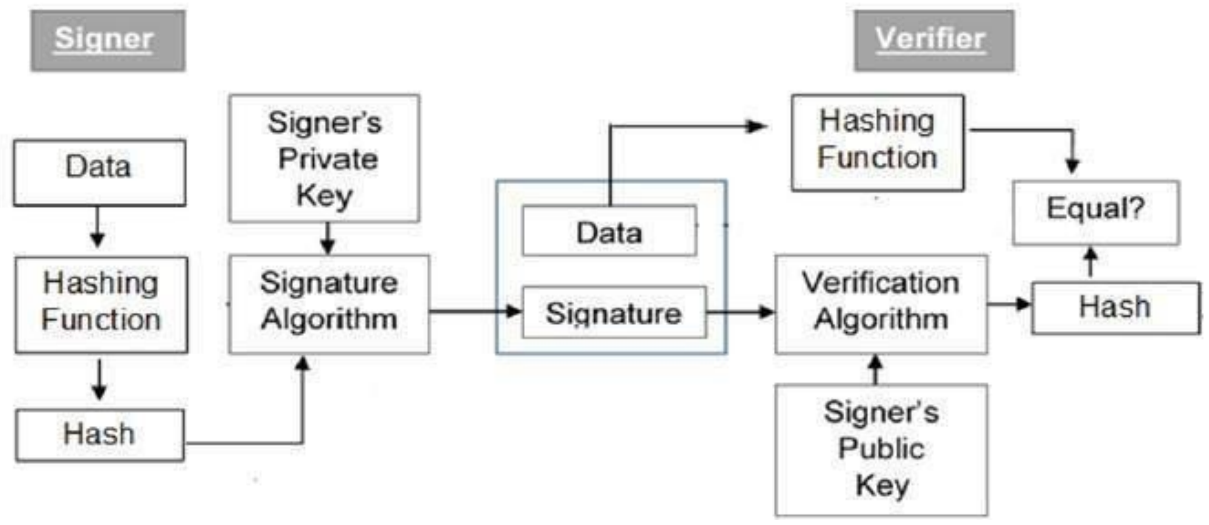
Practical: 10

Demonstrate working of Digital Signature using Cryptool

Description: A digital signature is a technique that binds a person/entity to the digital data. This binding can be independently verified by receiver as well as any third party.

Digital signature is a cryptographic value that is calculated from the data and a secret key known only by the signer.

The model of digital signature scheme is depicted in the following illustration − The model of digital signature scheme is depicted in the following illustration –



The following points explain the entire process in detail −

Each person adopting this scheme has a public-private key pair.

Generally, the key pairs used for encryption/decryption and signing/verifying are different. The private key used for signing is referred to as the signature key and the public key as the verification key.

Signer feeds data to the hash function and generates hash of data.

Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash. Signature is appended to the data and then both are sent to the verifier.

Verifier feeds the digital signature and the verification key into the verification algorithm. The verification algorithm gives some value as output.

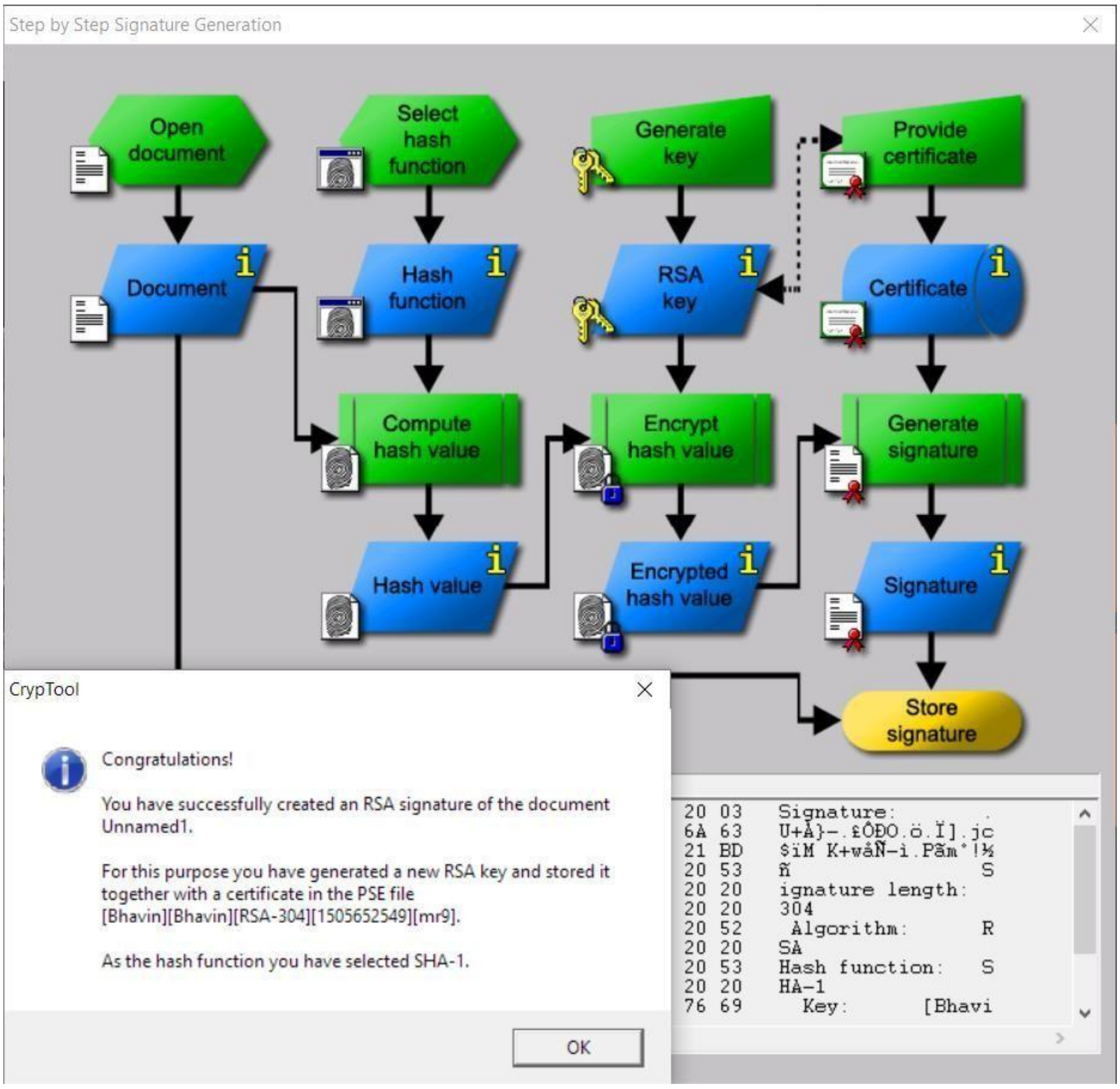
Verifier also runs same hash function on received data to generate hash value.

For verification, this hash value and output of verification algorithm are compared. Based on the comparison result, verifier decides whether the digital signature is valid.

Since digital signature is created by 'private' key of signer and no one else can have this key; the signer cannot repudiate signing the data in future.

Screenshot:

Implementation in Cryptool



Congratulations!

You have successfully created an RSA signature of the document Unnamed1.

For this purpose you have generated a new RSA key and stored it together with a certificate in the PSE file [Bhavin][Bhavin][RSA-304][1505652549][mr9].

As the hash function you have selected SHA-1.

```
20 03   Signature:         .
6A 63   U+Å}-.£ÔĐO.ö.Ï].jc
21 BD   $ïM K+wåÑ-i.Pãm°!½
20 53   ñ                  S
20 20   ignature length:
20 20   304
20 52    Algorithm:        R
20 20   SA
20 53   Hash function:     S
20 20   HA-1
76 69     Key:        [Bhavi
```