NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Visvesvaraya Technological University, Belgaum Approved by UGC, AICTE & Govt. of Karnataka)
Yelahanka, Bangalore-560064

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

LAB MANUAL

VI SEMESTER

Network Security Lab

Sub Code: 21ISL66

Prepared by:

Presilla R
Assistant Professor

Evangeline RC
Assistant Professor

Vani ES
Assistant Professor

Scheme:(2021)

TABLE OF CONTENTS

VISION AND MISSION OF THE DEPARTMENT

Vision

To achieve excellence in information science and engineering by empowering students with state-of-the-art technology and skills, inspiring them to drive innovation and entrepreneurship in addressing global challenges.

Mission

M1: Offer state-of-the-art curriculum through comprehensive pedagogical methods.

M2: Collaborate with industries and research institutions to address real-world challenges.

M3: Provide an environment that fosters creativity, critical thinking, innovation, and entrepreneurship.

Program Educational Objectives (PEOs)

- Graduates will progress in their careers in IT industries of repute.
- Graduates will succeed in higher studies and research.
- Graduates of Information Science and engineering will demonstrate highest integrity with ethical values, good communication skills, leadership qualities and self-learning abilities.

Program Outcomes

PO-1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. .
PO-2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
PO-3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
PO-4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO-5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO-6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO-7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO-8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
PO-9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO-10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO-11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO-12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Objectives (PSOs)

- Student will be able to understand the architecture and working of computer system with relevant system software and apply appropriate system calls.
- Student will be able to apply mathematical methodologies in modelling real world problems for the development of software applications using algorithms, data structures and programming tools.

INTRODUCTION AND SCOPE OF THE COURSE

Cryptography and Network Security (21IS63)

Cryptography

The art or science encompassing the principles and methods of transforming an intelligible message into one that is unintelligible, and then retransforming that message back to its original form

- **Plaintext** The original intelligible message
- **Cipher text** The transformed message
- **Cipher** An algorithm for transforming an intelligible message into one that is unintelligible by transposition and/or substitution methods
- **Key** Some critical information used by the cipher, known only to the sender& receiver
- **Encipher (encode)** The process of converting plaintext to cipher text using a cipher and a key
- **Decipher (decode)** the process of converting cipher text back into plaintext using a cipher and a key
- **Cryptanalysis** The study of principles and methods of transforming an unintelligible message back into an intelligible message without knowledge of the key. Also called code breaking
- **Cryptology** Both cryptography and cryptanalysis
- **Code** An algorithm for transforming an intelligible message into an unintelligible one using a code-book.

Department: *Information Science and Engineering*	***Course Type:*** *Core*
Course Title*: Network security Lab*	***Course Code:21ISL66***
L-T-P: *3-0-0*	***Credits:*** *01*
Total Contact Hours:	***Duration of SEE:*** *3 hrs*
SEE Marks*: 50*	***CIE Marks:*** *50*

Pre-requisites:

- Fundamental knowledge **Fundamentals of algorithm design techniques, Computer Networks.**

Course Outcomes:

Sl.no	Course outcomes	Blooms Level
1	Design and implement cipher techniques for encryption and decryption.	L3
2	Design and implement encryption and decryption algorithm ensuring confidentiality and integrity of data .	L3
3	Implement key exchange protocol for secure communication scenarios.	L3
4	Understand and analyze data encryption standard.	L3

Teaching Methodology:

- Black board Teaching
- PowerPoint Presentation

Assessment Methods:

- Two internals, 15Marks each will be conducted and the Average of two will be taken.
- 5M for viva-voce (Average of two internals)
- 30M for CIE (Rubrics)

Course Outcome:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	3	3	2	2				2	2		2		2
CO2	3	3	3	2	2				2	2		2		2
CO3	3	3	3	2	2				2	2		2		2
CO4	3	3	3	2	2				2	2		2		2
21ISL66	3	3	3	2	2				2	2		2		2

Course Content

1.	Imagine you're tasked with ensuring secure communication among field operatives in a covert operation. Design a C program implementing the Caesar Cipher algorithm to encrypt and decrypt messages exchanged between operatives, taking 'hello how are u' as the sample message and ensuring its confidentiality amidst potential interceptions
2.	Write a C program that encrypts a messages using a monoalphabetic cipher. Encrypt the plaintext 'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG' with the key 'QWERTYUIOPASDFGHJKLZXCVBNM' and print the resulting ciphertext. Then, decrypt the ciphertext 'QWE R TUIK BRGOS FXJVN TWHUZ SDG' using the same key and print the resulting plaintext."
3.	Develop a C program to perform encryption using the Playfair Cipher algorithm, where the user provides a key and a message to be encrypted and decrypted, facilitating secure communication in a network laboratory setting."
4.	Develop a C program for encryption using the Hill Cipher algorithm, allowing users to input a matrix key and a message to ensure secure communication."
5.	Develop a C program to encrypt and decrypt messages using the Polyalphabetic Cipher algorithm, employing the key 'deceptive' and the plaintext 'wearediscoveredsaveyourself'
6.	User A want to communicate to user B but they want to user Asymmetric Key Cryptography by using RSA algorithms send message to each other.encrypt message at sender side and decrypt it at receiver side
7.	Develop a mechanism to setup a security channel using Diffie-Hellman Key Exchange between client and server
8.	User C want to send message “welcome to ISE” to user D by using AES algorithms encrypt it and decrypt it at receiver end.

Lab evaluation rubrics matrix

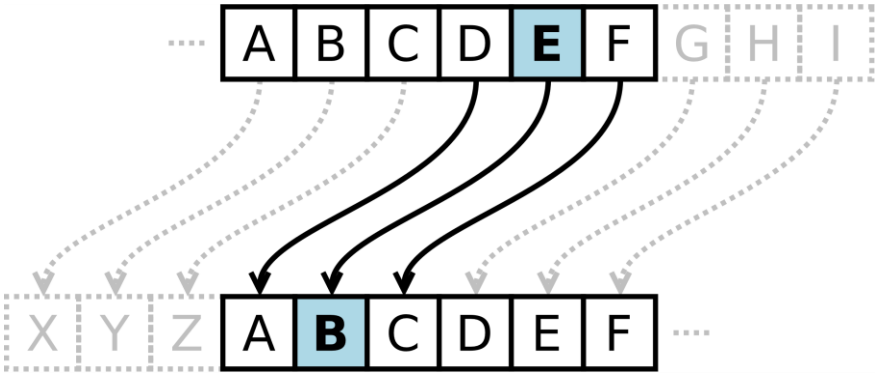
	Excellent(30)	Very good(25)	Good(20)	Average(10)
Understanding of problem statement	The problem statement is exceptionally clear, concise, and specific, leaving no room for ambiguity. **(10)**	The problem statement is clear and specific, but there may be minor areas of ambiguity. **(9)**	The problem statement lacks clarity and specificity, making it difficult to understand the exact nature of the problem. **(8)**	The problem statement is vague and lacks specificity, making it unclear what the problem actually entails **(3)**.
Flow of program logic	The problem statement follows a clear and logical sequence of ideas, facilitating easy understanding. **(10)**	The logical structure is mostly clear, with only minor instances of ambiguity or confusion. **(8)**	The logical structure is somewhat unclear or disjointed, requiring some effort to follow the flow of ideas. **(6)**	The problem statement lacks a coherent logical structure, making it difficult to understand the sequence of ideas. **(3)**
Generate Error free output	Output is completely error-free with no mistakes. **(5)**	Output is mostly error-free with minor, inconsequential mistakes, if any. **(4)**	Output contains several errors or inaccuracies, but they do not impede overall comprehension or functionality. **(3)**	Output is riddled with errors or inaccuracies, significantly impairing comprehension or functionality. **(2)**
Viva-Voce	Demonstrates a deep understanding of the subject matter, including theoretical concepts, practical applications, and related literature. **(5)**	Shows a solid understanding of the subject matter, covering key concepts and relevant information effectively. **(4)**	Demonstrates a basic understanding of the subject matter, but may struggle with more complex or nuanced aspects. **(3)**	Shows limited understanding of the subject matter, with significant gaps in knowledge or understanding. **(2)**

IMPLEMENTATION OF CAESAR CIPHER

AIM:

To implement the simple substitution technique named Caesar cipher using C language. DESCRIPTION: To encrypt a message with a Caesar cipher, each letter in the message is changed using a simple rule: shift by three. Each letter is replaced by the letter three letters ahead in the alphabet. A becomes D, B becomes E, and so on. For the last letters, we can think of alphabet as a circle and "wrap around". W becomes Z, X becomes A, Y becomes B, and Z becomes C. To change a message back, each letter is replaced by the one three before it.

EXAMPLE:



1)Imagine you're tasked with ensuring secure communication among field operatives in a covert operation. Design a C program implementing the Caesar Cipher algorithm to encrypt and decrypt messages exchanged between operatives, taking 'hello how are u' as the sample message and ensuring its confidentiality amidst potential interceptions.

ALGORITHM:

STEP-1: Read the plain text from the user.
STEP-2: Read the key value from the user.
STEP-3: If the key is positive then encrypt the text by adding the key with each character in the plain text.
STEP-4: Else subtract the key from the plain text.
STEP-5: Display the cipher text obtained above.

Program:

```
#include <stdio.h>
void caesar(char text[], int shift) {
   for (int i = 0; text[i] != '\0'; ++i) {
```

```
        if (text[i] >= 'a' && text[i] <= 'z')
            text[i] = 'a' + (text[i] - 'a' + shift) % 26;
        else if (text[i] >= 'A' && text[i] <= 'Z')
            text[i] = 'A' + (text[i] - 'A' + shift) % 26;
    }
}

int main() {
    char message[] = "hello how are u";
    int key = 3;

    printf("Original message: %s\n", message);

    // Encryption
    caesar(message, key);
    printf("Encrypted message: %s\n", message);

    // Decryption
    caesar(message, -key);
    printf("Decrypted message: %s\n", message);

    return 0;
}
```

OUTPUT:

Original message: hello how are u

Encrypted message: khoor krz duh x

Decrypted message: hello how are u

Monoalphabetic cipher

Aim:

The aim of a monoalphabetic cipher is to encrypt plaintext messages to prevent unauthorized individuals from understanding the content of the communication. It provides a basic level of confidentiality by substituting each letter with another letter from a fixed mapping.

Description:

In a monoalphabetic cipher, the mapping between the plaintext alphabet and the ciphertext alphabet remains constant throughout the encryption process. For example, if we shift each letter of the plaintext alphabet by three positions to the right, 'A' would become 'D', 'B' would become 'E', and so on. The same mapping applies to every occurrence of each letter in the plaintext.

2. Write a C program that encrypts and decrypts messages using a monoalphabetic cipher. Encrypt the plaintext 'THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG' with the key 'QWERTYUIOPASDFGHJKLZXCVBNM' and print the resulting ciphertext. Then, decrypt the ciphertext 'QWE R TUIK BRGOS FXJVN TWHUZ SDG' using the same key and print the resulting plaintext."

Algorithm:

1. Include necessary header files: `<stdio.h>` for standard input/output functions and `<string.h>` for string manipulation functions

2. Define the `encrypt` function:

   - Parameters: `plaintext` (array of characters) and `key` (array of characters).

   - Initialize an integer variable `i`

   - Declare a character array `ciphertext` to store the encrypted text, sized to accommodate the length of `plaintext` plus one for the null terminator.

   - Iterate over each character in `plaintext` using a loop.

   - If the character is an uppercase letter:

   - Encrypt it by substituting with the corresponding character from the `key`.

   - If the character is a lowercase letter:

   - Encrypt it similarly but using lowercase letters from the `key`.

   - Otherwise, leave the character unchanged.

   - Add a null terminator to `ciphertext`.

- Print the ciphertext.

3. Define the `decrypt` function:

- Parameters: `ciphertext` (array of characters) and `key` (array of characters).

- Initialize integer variables `i` and `j`.

- Declare a character array `plaintext` to store the decrypted text, sized to accommodate the length of `ciphertext` plus one for the null terminator.

- Iterate over each character in `ciphertext` using a loop.

- If the character is an uppercase letter:

- Decrypt it by finding its position in the `key` and mapping it back to an uppercase letter.

- If the character is a lowercase letter:

- Decrypt it similarly but considering lowercase letters from the `key`.

- Otherwise, leave the character unchanged.

- Add a null terminator to `plaintext`.

- Print the decrypted plaintext.

4. Define the `main` function:

- Initialize character arrays `plaintext` and `key` with the original plaintext and the encryption key, respectively.

- Print the original plaintext.

- Call the `encrypt` function with `plaintext` and `key`.

- Initialize a character array `ciphertext` with a sample ciphertext.

- Call the `decrypt` function with `ciphertext` and `key`.

- Return 0 to indicate successful program execution.

5. Print statements within `main` display intermediate and final results.

6. Execute the program to demonstrate encryption and decryption of text using a monoalphabetic cipher.

Program:

```
#include <stdio.h>
#include <string.h>
// Function to encrypt plaintext using monoalphabetic cipher
void encrypt(char plaintext[], char key[]) {
    int i;
    char ciphertext[strlen(plaintext) + 1];
    for (i = 0; plaintext[i] != '\0'; i++) {
        if (plaintext[i] >= 'A' && plaintext[i] <= 'Z') {
            ciphertext[i] = key[plaintext[i] - 'A'];
        } else if (plaintext[i] >= 'a' && plaintext[i] <= 'z') {
            ciphertext[i] = key[plaintext[i] - 'a'];
        } else {
            ciphertext[i] = plaintext[i];
        }
    }
    ciphertext[i] = '\0';
    printf("Ciphertext: %s\n", ciphertext);
}
// Function to decrypt ciphertext using monoalphabetic cipher

void decrypt(char ciphertext[], char key[]) {
    int i, j;
    char plaintext[strlen(ciphertext) + 1];
    for (i = 0; ciphertext[i] != '\0'; i++) {
        if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {
            for (j = 0; key[j] != '\0'; j++) {
                if (ciphertext[i] == key[j]) {
                    plaintext[i] = 'A' + j;
                    break;
                }
            }
        } else if (ciphertext[i] >= 'a' && ciphertext[i] <= 'z') {
            for (j = 0; key[j] != '\0'; j++) {
                if (ciphertext[i] == key[j] + 'a' - 'A') {
                    plaintext[i] = 'a' + j;
                    break;
                }
            }
        } else {
            plaintext[i] = ciphertext[i];
        }
    }
    plaintext[i] = '\0';
```

```
    printf("Decrypted plaintext: %s\n", plaintext);
}
     int main()
 {
    char plaintext[] = "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG";
    char key[] = "QWERTYUIOPASDFGHJKLZXCVBNM";
    printf("Plaintext: %s\n", plaintext);
    encrypt(plaintext, key);
    char ciphertext[] = "ZIT JXOEA WKGVF YGB PXDHL GCTK ZIT SQMN RGU";
    // printf("Ciphertext: %s\n", ciphertext);
    decrypt(ciphertext, key);
    return 0;
}
```

OUTPUT:

Plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

Ciphertext: ZIT JXOEA WKGVF YGB PXDHL GCTK ZIT SQMN RGU

Decrypted plaintext: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

IMPLEMENTATION OF PLAYFAIR CIPHER

AIM: To write a C program to implement the Playfair Substitution technique.

DESCRIPTION: The Playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet is omitted from the table (as there are 25 spots and 26 letters in the alphabet).

To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. The two letters of the diagram are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter

2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively

3.If the letters appear on the same column of your table, replace them with the letters immediately below respectively

4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.

EXAMPLE:

D. Playfair Cipher

Example1: Plaintext: CRYPTO IS TOO EASY **Key =** INFOSEC **Ciphertext:** ??

Grouped text: CR YP TO IS TO XO EA SY

Ciphertext: AQ TV YB NI YB YF CB OZ

I / J	N	F	O	S
E	C	A	B	D
G	H	K	L	M
P	Q	R	T	U
V	W	X	Y	Z

3. Develop a C program to perform encryption using the Playfair Cipher algorithm, where the user provides a key and a message to be encrypted, facilitating secure communication in a network laboratory setting.

ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the keyword from the user.

STEP-3: Arrange the keyword without duplicates in a 5*5 matrix in the row order and fill the remaining cells with missed out letters in alphabetical order. Note that ‘i’ and ‘j’ takes the same cell.

STEP-4: Group the plain text in pairs and match the corresponding corner letters by forming a rectangular grid.

STEP-5: Display the obtained cipher text.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/* Function for Playfair Cipher encryption  */
void Playfair(char str[], char keystr[]) {
   char keyMat[5][5];
   // Key & plainText
   char ks = strlen(keystr);
   char ps = strlen(str);
   void toUpperCase(char encrypt[], int ps) {
```

```
    for (int i= 0; i < ps; i++) {
        if (encrypt[i] > 96 && encrypt[i] < 123)
            encrypt[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;
}
/* this function will create a 5 by 5 matrix. */
void createMatrix(char keystr[], int ks, char keyMat[5][5]) {
    int flag = 0, *dict;
    /* here we are creating a hashmap for alphabets */
    dict = (int*)calloc(26, sizeof(int));
    for (int i = 0; i < ks; i++) {
        if (keystr[i] != 'j')
            dict[keystr[i] - 97] = 2;
    }
    dict['j' - 97] = 1;
    int i = 0, j = 0;
    for (int 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 (int k = 0; k < 26; k++) {
        if (dict[k] == 0) {
            keyMat[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}
```

```
/*this function looks for a digraph's characters in the key matrix and returns their positions.*/
void search(char keyMat[5][5], char a, char b, int arr[]) {
    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';
    for(int i = 0; i < 5; i++) {
        for(int 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;
            }
        }
    }
}
/* This function avoids duplication and levels out the length of plain text by making it even.*/
int prep(char str[], int p) {
    int sub = p;
    for (int i = 0; i < sub; i += 2) {
        if(str[i]==str[i+1]){
            for(int 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;
}
// Here, the encryption is done.
void encrypt(char str[], char keyMat[5][5], int pos) {
    int a[4];
    for(int 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]];
            }
        }
    }
    ks = removeSpaces(keystr, ks);
    ps = removeSpaces(str, ps);
    ps = prep(str, ps);
    createMatrix(keystr, ks, keyMat);
    encrypt(str, keyMat, ps);
    toUpperCase(str, ps);
    /* str is the final encrypted string in uppercase */
    printf("Cipher 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 Cipher Program in C functional call
    Playfair(string, keyString);
    return 0;
}
```

OUTPUT:

Enter key: war

Enter plaintext: hello

Cipher text: DFQRIQ

IMPLEMENTATION OF HILL CIPHER

AIM: To write a C program to implement the hill cipher substitution techniques.

DESCRIPTION:

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 is multiplied by an invertible $n \times n$ 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 $n \times n$ matrices (modulo 26).

EXAMPLE:

$$\begin{bmatrix} 2 & 4 & 5 \\ 9 & 2 & 1 \\ 3 & 17 & 7 \end{bmatrix} \begin{bmatrix} 0 \\ 19 \\ 19 \end{bmatrix} = \begin{bmatrix} 171 \\ 57 \\ 456 \end{bmatrix} \pmod{26} = \begin{bmatrix} 15 \\ 5 \\ 14 \end{bmatrix} = \text{'PFO'}$$

4. Develop a C program for encryption using the Hill Cipher algorithm, allowing users to input a matrix key and a message to ensure secure communication."

ALGORITHM:

STEP-1: Read the plain text and key from the user.
STEP-2: Split the plain text into groups of length three.
STEP-3: Arrange the keyword in a 3*3 matrix.
STEP-4: Multiply the two matrices to obtain the cipher text of length three.
STEP-5: Combine all these groups to get the complete cipher text.

Program:

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

#define MAX_SIZE 10
```

```
// Function to calculate the determinant of a 2x2 matrix
int determinant(int a, int b, int c, int d) {
    return (a * d - b * c);
}

// Function to calculate the modulo inverse of a number
int moduloInverse(int a, int m) {
    int i;
    for (i = 1; i < m; i++) {
        if ((a * i) % m == 1) {
            return i;
        }
    }
    return -1; // No modular inverse exists
}

// Function to encrypt the message using the Hill Cipher algorithm
void encrypt(char message[], int keyMatrix[][MAX_SIZE], int keySize) {
    int messageLength = strlen(message);
    int paddedLength = (int) ceil(messageLength / (float) keySize) * keySize;
    int paddedMessage[paddedLength];
    int encrypted[paddedLength];
    int i, j, k, sum;

    // Pad the message with 'X' to make its length a multiple of keySize
    for (i = 0; i < messageLength; i++) {
        paddedMessage[i] = message[i] - 'A';
    }
    for (; i < paddedLength; i++) {
        paddedMessage[i] = 23; // 'X' - 'A' = 23
    }

    // Encrypt the message
    for (i = 0; i < paddedLength; i += keySize) {
        for (j = 0; j < keySize; j++) {
            sum = 0;
            for (k = 0; k < keySize; k++) {
                sum += keyMatrix[j][k] * paddedMessage[i + k];
            }
            encrypted[i + j] = sum % 26;
        }
    }

    // Print the encrypted message
    printf("Encrypted message: ");
    for (i = 0; i < paddedLength; i++) {
```

```
        printf("%c", (char) (encrypted[i] + 'A'));
    }
    printf("\n");
}

int main() {
    int keySize, keyMatrix[MAX_SIZE][MAX_SIZE], i, j;
    char message[1000];

    // Input the size of the key matrix
    printf("Enter the size of the key matrix (max 10): ");
    scanf("%d", &keySize);

    // Input the key matrix elements
    printf("Enter the elements of the key matrix:\n");
    for (i = 0; i < keySize; i++) {
        for (j = 0; j < keySize; j++) {
            scanf("%d", &keyMatrix[i][j]);
        }
    }

    // Check if the key matrix is invertible
    if (determinant(keyMatrix[0][0], keyMatrix[0][1], keyMatrix[1][0], keyMatrix[1][1]) == 0) {
        printf("The key matrix is not invertible. Please enter a valid key matrix.\n");
        return 1;
    }

    // Input the message to be encrypted
    printf("Enter the message to be encrypted (uppercase letters only): ");
    scanf("%s", message);

    // Encrypt the message using the Hill Cipher algorithm
    encrypt(message, keyMatrix, keySize);

    return 0;
}
```

OUTPUT:

```
Enter the size of the key matrix (max 10): 2

Enter the elements of the key matrix:
1 2
3 4
Enter the message to be encrypted (uppercase letters only): HELLO
Encrypted message: PLHZIE
```

5. Develop a C program to encrypt and decrypt messages using the Polyalphabetic Cipher algorithm, employing the key 'deceptive' and the plaintext 'weareddiscoveredsaveyourself'.

Aim:
Develop a C program to implement encryption and decryption using the Polyalphabetic Cipher algorithm with a given key and plaintext.

Description:
The Polyalphabetic Cipher is a type of substitution cipher where multiple cipher alphabets are used to encrypt the plaintext. The key is a keyword or phrase that determines the shifting pattern for each letter in the plaintext. In this problem, we'll be implementing the Polyalphabetic Cipher algorithm with the key 'deceptive' and encrypting the plaintext 'weareddiscoveredsaveyourself'.

Example:
Given:
Plaintext: weareddiscoveredsaveyourself
Key: deceptive
Encryption:

Repeat the key until it matches the length of the plaintext:
Key: deceptive deceptive deceptive deceptived
Plaintext: weareddiscoveredsaveyourself
Encrypt each letter in the plaintext using the corresponding letter in the repeated key:
w (plaintext) + d (key) = z
e (plaintext) + e (key) = x
a (plaintext) + c (key) = c
r (plaintext) + e (key) = x
...
Repeat steps 1-2 for the entire plaintext.

Algorithm:

1. Start with the given plaintext and key.
2. Repeat the key until it matches the length of the plaintext.
3. Encrypt or decrypt each letter in the plaintext using the corresponding letter in the repeated key:
    a. For encryption: Add the numerical values of the plaintext letter and key letter, subtract 97 (the ASCII value of 'a'), and take the remainder when divided by 26. Convert the result back to a letter.
    b. For decryption: Subtract the numerical values of the key letter from the ciphertext letter, add 26 if the result is negative, and convert the result back to a letter.
4. Output the resulting ciphertext or decrypted plaintext.

This algorithm ensures that each letter in the plaintext is encrypted or decrypted using a different shifting pattern

based on the corresponding letter in the key, providing stronger security compared to simple substitution ciphers.

Program:

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

#define MAX_LEN 100

int main() {
    char pt[MAX_LEN] = {'\0'}, ct[MAX_LEN] = {'\0'}, key[MAX_LEN] = {'\0'}, rt[MAX_LEN] = {'\0'};
    int i, j;

    system("cls");  // Clear screen (for Windows)

    printf("\nEnter the plain text: ");
    fgets(pt, MAX_LEN, stdin);  // Safer input handling

    printf("Enter the key: ");
    fgets(key, MAX_LEN, stdin);

    // Remove newline characters from inputs
    pt[strcspn(pt, "\n")] = '\0';
    key[strcspn(key, "\n")] = '\0';

    // Ensure key is not empty
    if (strlen(key) == 0) {
        printf("Key cannot be empty.\n");
        return 1;
    }

    // Length of plaintext equal to or greater than length of key
    j = 0;
    for (i = strlen(key); i < strlen(pt); i++) {
        key[i] = key[j % strlen(key)];  // Cycling through the key characters
        j++;
    }
    printf("\nNew key is: %s\n", key);

    // Encrypt plaintext
    for (i = 0; i < strlen(pt); i++) {
        ct[i] = (((pt[i] - 'a') + (key[i] - 'a')) % 26) + 'a';  // Ensure modulo operation works correctly
    }
    printf("\nCipher text is: %s\n", ct);

    // Decrypt ciphertext
    for (i = 0; i < strlen(ct); i++) {
```

```
        if (ct[i] < key[i % strlen(key)]) {
            rt[i] = 26 + ((ct[i] - 'a') - (key[i % strlen(key)] - 'a')) + 'a';
        } else {
            rt[i] = (((ct[i] - 'a') - (key[i % strlen(key)] - 'a')) % 26) + 'a';
        }
    }
    printf("\nPlain text is: %s\n", rt);

    return 0;
}
```

Output:

Enter the key: deceptive

New key is: deceptivedeceptivedeceptive
Cipher text is: zicvtwqngrzgvtwavzhcqyglmgj
Plain text is: wearediscoveredsaveyourself

6. User A want to communicate to user B but they want to user Asymmetric Key Cryptography by using RSA algorithms send message to each other encrypt message at sender side and decrypt it at receiver side

Aim: Implement RSA asymmetric key cryptography to enable secure communication between User A and User B, allowing them to encrypt messages at the sender side and decrypt them at the receiver side.

Description: RSA (Rivest-Shamir-Adleman) is a widely-used asymmetric encryption algorithm for secure communication. It involves the use of a public key for encryption and a private key for decryption. In this scenario, User A will use User B's public key to encrypt the message, and User B will use their private key to decrypt the message.

Algorithm:

1. **Key Generation**:
    - User B generates a pair of RSA keys: a public key (e, n) and a private key (d, n).
    - The public key (e, n) is shared with User A, while the private key (d, n) is kept secret by User B.
2. **Encryption (Sender Side)**:
    - User A obtains User B's public key (e, n).
    - User A converts the plaintext message into an integer m, where m < n.
    - User A computes the ciphertext c using the RSA encryption formula:

      $c = m^e \bmod n$.
    - User A sends the ciphertext c to User B.

3. **Decryption (Receiver Side)**:
    - User B receives the ciphertext c.
    - User B uses their private key (d, n) to compute the original message m using the RSA decryption formula: $m=c^d \bmod n$.
    - User B converts the integer m back into the plaintext message.

Example:

Let's say User B generates RSA keys with the following parameters:

- Public key (e, n): (17, 3233)
- Private key (d, n): (2753, 3233)

User A wants to send the message "HELLO" to User B:

1. User A converts "HELLO" into an integer m.
2. User A encrypts m using User B's public key: $c=m^{17} \bmod 3233$
3. User A sends the ciphertext c to User B.
4. User B decrypts the ciphertext c using their private key:$m=c^{2753} \bmod 3233$.
5. User B converts the integer m back into the plaintext message "HELLO".

This process ensures that only User B can decrypt the message using their private key, providing secure communication between the two users.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>

#define MAX_LEN 1024

int main() {
   RSA *rsa_A = NULL, *rsa_B = NULL;
   unsigned char *plaintext = (unsigned char *)"Hello, User B!";
   unsigned char ciphertext[MAX_LEN] = {0};
   unsigned char decryptedtext[MAX_LEN] = {0};

   // Key Generation for User A
```

```
    rsa_A = RSA_generate_key(2048, RSA_F4, NULL, NULL);
    if (rsa_A == NULL) {
        fprintf(stderr, "Error generating RSA key for User A\n");
        return 1;
    }

    // Key Generation for User B
    rsa_B = RSA_generate_key(2048, RSA_F4, NULL, NULL);
    if (rsa_B == NULL) {
        fprintf(stderr, "Error generating RSA key for User B\n");
        RSA_free(rsa_A);
        return 1;
    }

    // Encryption by User A
    int encrypted_len = RSA_public_encrypt(strlen((char *)plaintext), plaintext, ciphertext, rsa_B,
RSA_PKCS1_PADDING);
    if (encrypted_len == -1) {
        fprintf(stderr, "Error encrypting message\n");
        RSA_free(rsa_A);
        RSA_free(rsa_B);
        return 1;
    }

    // Decryption by User B
    int decrypted_len = RSA_private_decrypt(encrypted_len, ciphertext, decryptedtext, rsa_B,
RSA_PKCS1_PADDING);
    if (decrypted_len == -1) {
        fprintf(stderr, "Error decrypting message\n");
        RSA_free(rsa_A);
        RSA_free(rsa_B);
        return 1;
    }

    printf("Encrypted message by User A: ");
    for (int i = 0; i < encrypted_len; i++) {
        printf("%02x", ciphertext[i]);
    }
    printf("\n");

    printf("Decrypted message by User B: %s\n", decryptedtext);

    RSA_free(rsa_A);
    RSA_free(rsa_B);

    return 0;
```

```
}
```

Compile:

```
gcc -o rsa_example rsa_example.c -lssl –lcrypto
```

Output:

Encrypted message by User A: 75e5aa857a47e204c7d0c09c70d79ee8bf240051de...
Decrypted message by User B: Hello, User B!

[or]

```
#include <stdio.h>

// Function to calculate (base^exp) % modulus
long long int power(long long int base, long long int exp, long long int modulus) {
    long long int result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exp /= 2;
    }
    return result;
}

// Function to perform RSA encryption
long long int encrypt(long long int plaintext, long long int e, long long int n) {
    return power(plaintext, e, n);
}

// Function to perform RSA decryption
long long int decrypt(long long int ciphertext, long long int d, long long int n) {
    return power(ciphertext, d, n);
}

int main() {
    // Public and private keys
    long long int p = 61;
    long long int q = 53;
    long long int n = p * q;
    long long int phi = (p - 1) * (q - 1);
    long long int e = 17; // Public exponent
    long long int d = 413; // Private exponent

    // Message to be encrypted
    long long int plaintext = 123;
```

```
    // Encryption
    long long int ciphertext = encrypt(plaintext, e, n);
    printf("Encrypted message: %lld\n", ciphertext);

    // Decryption
    long long int decryptedtext = decrypt(ciphertext, d, n);
    printf("Decrypted message: %lld\n", decryptedtext);

    return 0;
}
```

Output:
Encrypted message: 1692
Decrypted message: 123

7. Develop a mechanism to setup a security channel using Diffie-Hellman Key Exchange between client and server

Aim: Develop a mechanism to establish a secure communication channel between a client and a server using the Diffie-Hellman Key Exchange algorithm.

Description: The Diffie-Hellman Key Exchange algorithm allows two parties to establish a shared secret key over an insecure communication channel without exchanging the key explicitly. This shared key can then be used for encryption and decryption, ensuring secure communication between the parties.

Mechanism:

1. **Setup**:
    - Both the client and the server agree on two public parameters: a large prime number p and a primitive root modulo g.
    - These parameters are typically chosen beforehand and are public knowledge.
2. **Key Generation**:
    - Both the client and the server generate their private keys:
        - Client: Choose a random integer a (private key).
        - Server: Choose a random integer b (private key).
    - Each party calculates their public key using the formula: $A=g^a \bmod p$ or $B=g^b \bmod p$.
    - The public keys A and B are exchanged between the client and the server.
3. **Shared Secret Calculation**:

- Once the public keys are exchanged, each party can calculate the shared secret key:
  - Client: $S=B^a \bmod p$
  - Server: $S=A^b \bmod p$
- Both the client and the server now have the same shared secret key S, which can be used for encryption and decryption.

4. **Encryption and Decryption**:
   - After establishing the shared secret key, the client and the server can use symmetric encryption algorithms (e.g., AES) with the shared key S to encrypt and decrypt messages exchanged between them.

Example: Let's illustrate the Diffie-Hellman Key Exchange mechanism with a simple example:

1. Client and server agree on public parameters: $p=23$ and $g=5$.
2. Client chooses $a=6$ (private key), calculates $A=5^6 \bmod 23=8$, and sends $A=8$ to the server.
3. Server chooses $b=15$ (private key), calculates $B=5^{15} \bmod 23=19$, and sends $B=19$ to the client.
4. Client calculates shared secret $S=19^6 \bmod 23=2$.
5. Server calculates shared secret $S=8^{15} \bmod 23=2$.
6. Both client and server now have the shared secret $S=2$ and can use it for encryption and decryption.

This mechanism ensures that even if an eavesdropper intercepts the public keys exchanged between the client and the server, they cannot determine the shared secret key without knowing the private keys, providing secure communication between the two parties.

Program:

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

#define P 23 // Prime number
#define G 5  // Primitive root modulo

// Function to calculate modular exponentiation (base^exp % modulus)
int mod_exp(int base, int exp, int modulus) {
    int result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
```

```
        exp /= 2;
    }
    return result;
}

// Function to perform Diffie-Hellman Key Exchange

void diffie_hellman(int private_key, int *public_key) {
    *public_key = mod_exp(G, private_key, P);
}

int main() {
    int private_key_client, private_key_server;
    int public_key_client, public_key_server;
    int shared_secret_client, shared_secret_server;

    // Generate private keys for client and server
    private_key_client = rand() % (P - 1) + 1;
    private_key_server = rand() % (P - 1) + 1;

    // Perform Diffie-Hellman Key Exchange for client
    diffie_hellman(private_key_client, &public_key_client);
    // Perform Diffie-Hellman Key Exchange for server
    diffie_hellman(private_key_server, &public_key_server);

    // Calculate shared secrets
    shared_secret_client = mod_exp(public_key_server, private_key_client, P);
    shared_secret_server = mod_exp(public_key_client, private_key_server, P);

    // Print shared secrets
    printf("Client shared secret: %d\n", shared_secret_client);
    printf("Server shared secret: %d\n", shared_secret_server);

    return 0;
}
```

Output:

Client shared secret: 1
Server shared secret: 1

8. User C want to send message “welcome to ISE” to user D by using AES algorithms encrypt it and decrypt it at receiver end.

Aim: Implement the AES encryption algorithm to securely send a message from User C to User D and decrypt it at the receiver end.

Description: AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely used for securing sensitive data. It operates on fixed-size blocks of data and supports key lengths of 128, 192, or 256 bits. In this scenario, User C will encrypt the message "welcome to ISE" using AES encryption and send it to User D, who will decrypt the message using the same key.

Mechanism:

1. **Encryption (Sender Side)**:
    - User C obtains the plaintext message "welcome to ISE".
    - User C selects a secret key (128, 192, or 256 bits) for AES encryption.
    - User C encrypts the plaintext message using the AES encryption algorithm and the secret key to obtain the ciphertext.
    - User C sends the ciphertext to User D over the communication channel.
2. **Decryption (Receiver Side)**:
    - User D receives the ciphertext from User C.
    - User D uses the same secret key used by User C to encrypt the message.
    - User D decrypts the ciphertext using the AES decryption algorithm and the secret key to obtain the original plaintext message.

Example:

Let's illustrate the AES encryption and decryption process using a simple example:

- Plaintext message: "welcome to ISE"
- Secret key: "mysecretkey12345" (128-bit key)

1. Encryption (Sender Side):
    - User C encrypts the plaintext message "welcome to ISE" using AES encryption with the secret key "mysecretkey12345".
    - The ciphertext is generated: e.g., "47ec6f5b3c9db2b375e5f7e4cd96221c".
2. Decryption (Receiver Side):
    - User D receives the ciphertext "47ec6f5b3c9db2b375e5f7e4cd96221c" from User C.

- User D decrypts the ciphertext using AES decryption with the same secret key "mysecretkey12345".
- The original plaintext message "welcome to ISE" is recovered.

Note: In practice, secure key exchange mechanisms (e.g., Diffie-Hellman) may be used to securely share the secret key between User C and User D before encryption and decryption. Additionally, padding schemes may be used to ensure that the plaintext message length is compatible with the block size of AES.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/aes.h>

// AES key for encryption and decryption (128-bit key)
unsigned char aes_key[] = "0123456789abcdef";

// Function to encrypt plaintext using AES
void aes_encrypt(unsigned char *plaintext, unsigned char *ciphertext) {
    AES_KEY key;
    AES_set_encrypt_key(aes_key, 128, &key);
    AES_encrypt(plaintext, ciphertext, &key);
}

// Function to decrypt ciphertext using AES
void aes_decrypt(unsigned char *ciphertext, unsigned char *decryptedtext) {
    AES_KEY key;
    AES_set_decrypt_key(aes_key, 128, &key);
    AES_decrypt(ciphertext, decryptedtext, &key);
}

int main() {
    unsigned char plaintext[] = "welcome to ISE";
    unsigned char ciphertext[AES_BLOCK_SIZE]; // AES block size is 128 bits
    unsigned char decryptedtext[AES_BLOCK_SIZE];

    // Encrypt the plaintext
    aes_encrypt(plaintext, ciphertext);

    // Decrypt the ciphertext
    aes_decrypt(ciphertext, decryptedtext);

    printf("Original message: %s\n", plaintext);
    printf("Encrypted message: ");
    for (int i = 0; i < AES_BLOCK_SIZE; i++) {
```

```
        printf("%02x", ciphertext[i]);
    }
    printf("\n");
    printf("Decrypted message: %s\n", decryptedtext);

    return 0;
}
```

Output:

Original message: welcome to ISE

Encrypted message: d9c4c8a23f773ebc5b37904164f38b2a

Decrypted message: welcome to ISE

Viva Questions

1. What is network security, and why is it important in modern computing?
2. Explain the CIA triad and its significance in network security.
3. How would you differentiate between authentication and authorization?
4. Define symmetric and asymmetric encryption algorithms and provide examples of each.
5. Can you explain how a digital signature functions and its relevance in network security?
6. Describe the purpose of cryptographic hash functions in network security.
7. Discuss common network-layer attacks like DoS and DDoS attacks.
8. What is ARP spoofing, and what measures can be taken to prevent it?
9. Explain how SSL/TLS protocols secure communication over the internet.
10. What role do firewalls play in network security, and what are the different types of firewalls?
11. How does an Intrusion Detection System (IDS) differ from a firewall, and what are its key components?
12. Can you explain how symmetric encryption works and provide an example of a symmetric encryption algorithm?
13. What are some advantages and disadvantages of using symmetric encryption in network security?
14. How does the process of key distribution work in symmetric encryption systems?
15. Describe the principles behind asymmetric encryption and provide an example of an asymmetric encryption algorithm.
16. What role do public and private keys play in asymmetric encryption, and how are they generated?
17. Discuss the advantages and limitations of asymmetric encryption compared to symmetric encryption.
18. Explain the purpose of cryptographic hash functions in network security.
19. What properties should a secure hash function possess?
20. How are hash functions used in digital signatures and data integrity verification?
21. Define digital signatures and their significance in ensuring message authenticity and integrity.
22. How does a digital signature scheme work, and what role do hash functions play in this process?
23. What are some potential vulnerabilities associated with digital signatures, and how can they be mitigated?
24. Describe the purpose of key exchange algorithms in establishing secure communication channels.

25. Explain the Diffie-Hellman key exchange algorithm and its significance in secure communication protocols.
26. How does the RSA algorithm facilitate key exchange in asymmetric encryption systems?
27. What are block cipher modes of operation, and why are they important in symmetric encryption?
28. Describe the Electronic Codebook (ECB) and Cipher Block Chaining (CBC) modes, including their strengths and weaknesses.
29. How does the initialization vector (IV) enhance the security of block cipher modes like CBC?
30. Define stream ciphers and provide an example of a stream cipher algorithm.
31. How do stream ciphers differ from block ciphers in terms of operation and application?
32. Discuss the advantages and limitations of stream ciphers compared to block ciphers.
33. What is cryptanalysis, and how does it relate to security algorithms?
34. Describe some common cryptanalysis techniques used to break encryption schemes.
35. How can encryption algorithms be strengthened against cryptanalysis attacks?

REFERENCES

Cryptography and Network Security: Principles and Practice by William Stallings
Applied Cryptography: Protocols, Algorithms, and Source Code in C by Bruce Schneier
Cryptography Engineering: Design Principles and Practical Applications by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno.
GitHub Repositories