Date: 10/ 12 / 2024

Lab Practical #01:

Write a Prerequisite Programs in any language. (C/Java)

1. Calculate the swapping of two numbers using function.
2. Check if a number is prime.
3. Write a program with a function to calculate factorial.
4. Write a program to reverse a string using pointers.
5. Implement string functions like strlen, strcat, and strcmp.
6. Create a program to store and retrieve student details using struct.
7. Implement a program to read and write a text file.
8. Implement a program to check if a number is a power of 2 using bitwise operators.
9. Create a linked list or a dynamic array.
10. Write a program to encode a decimal into binary and vice versa.
11. Write a client-server chat application.
12. Implement file transfer over a network.

Date: 10/ 12 / 2024

1. Calculate the swapping of two numbers using function.

```
#include <stdio.h>

void swapa(int a, int b)

{

   a = a + b;

   b = a - b;

   a = a - b;

   printf("After swapping: x = %d, y = %d\n", a, b);

}

void main()

{

   int x, y;

   printf("Enter two number: ");

   scanf("%d %d", &x, &y);

   printf("Before swapping: x = %d, y = %d\n", x, y);

   swapa(x, y);

}
```

Date: 10/ 12 / 2024

Output:

```
Enter two number: 5

6

Before swapping: x = 5, y = 6

After swapping: x = 6, y = 5
```

2 Check if a number is prime.

```
#include<stdio.h>

int main(){

   int a,i;

   int flag=0;

   printf("Enter a number: ");

   scanf("%d", &a);

   for(i=2; i<=a/2; i++){

      if(a % i == 0){

         flag = 1;

         break;

      }

   }

   if(flag == 0)

      printf("%d is a prime number.\n", a);

   else

      printf("%d is not a prime number.\n", a);

}
```

Date: 10/ 12 / 2024

Output:

```
Enter a number: 5

5 is a prime number.
```

3.Write a program with a function to calculate factorial.

```
#include <stdio.h>

int factorial(int n) {

   if (n == 0) {

      return 1;

   }

   return n * factorial(n - 1);

}

int main() {

   int n;

   printf("Enter a number: ");

   scanf("%d", &n);

   printf("Factorial of %d is %d\n", n, factorial(n));

   return 0;

}
```

Output:

```
Enter a number: 5

Factorial of 5 is 120
```

Date: 10/ 12 / 2024

4. Write a program to reverse a string using pointers.

```
#include <stdio.h>
#include <string.h>
void main(){
   char str[100];
   printf("Enter a string:");
   scanf("%s", str);
   int length=strlen(str);
   char *start = str;
   char *end = str + length - 1;
   while (start < end)
   {
      char temp = *start;
      *start = *end;
      *end = temp;
      start++;
      end--;
   }
   printf("The reverse of the string is \"%s\".\n", str);
}
```

Output:

```
Enter a string:nirav
The reverse of the string is "varin".
```

Date: 10/ 12 / 2024

5. Implement string functions like strlen, strcat, and strcmp.

```
#include <stdio.h>

#include <string.h>


int main() {

   const char *str = "Hello, World!";

   printf("Length of string: %zu\n", strlen(str));


   const char *str1 = "Hello";

   const char *str2 = "World";

   int cmp_result = strcmp(str1, str2);

   printf("Comparison result: %d\n", cmp_result);


   char dest[50] = "Hello, ";

   const char *src = "World!";

   strcat(dest, src);

   printf("Concatenated string: %s\n", dest);


   return 0;

}
```

Output:

```
Length of string: 13

Comparison result: -1

Concatenated string: Hello, World!
```

Date: 10/ 12 / 2024

6. Create a program to store and retrieve student details using struct.

```
#include <stdio.h>

#include <string.h>

struct Student {

    int id;

    char name[50];

    float grade;

};

void addStudent(struct Student *s, int id, const char *name, float grade) {

    s->id = id;

    strcpy(s->name, name);

    s->grade = grade;

}

void displayStudent(struct Student s) {

    printf("ID: %d\n", s.id);

    printf("Name: %s\n", s.name);

    printf("Grade: %.2f\n", s.grade);

}

int main() {

    struct Student student1;

    addStudent(&student1, 1, "nirav", 85.5);

    displayStudent(student1);

    return 0;

}
```

Date: 10/ 12 / 2024

Output:

```
ID: 1

Name: nirav

Grade: 85.50
```

7. Implement a program to read and write a text file.

```
#include <stdio.h>

int main() {

   const char *filename = "example.txt";

   FILE *file;

   char content[255];

   file = fopen(filename, "w");

   if (file == NULL) {

      printf("Error opening file!\n");

      return 1;

   }

   printf("Please enter content to write to the text file: ");

   fgets(content, sizeof(content), stdin);

   fprintf(file, "%s", content);

   fclose(file);

   printf("Content written to '%s'.\n", filename);

   file = fopen(filename, "r");
```

Date: 10/ 12 / 2024

```
if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    printf("File content:\n");
    while (fgets(content, sizeof(content), file) != NULL) {
        printf("%s", content);
    }
    fclose(file);
    return 0;
}
```

note:after run this code in this file directory create welcome.txt file and write in a file.

Output:

```
Please enter content to write to the text file: hi welcome
Content written to 'example.txt'.
File content:
hi welcome
```

Date: 10/ 12 / 2024

8.Implement a program to check if a number is a power of 2 using bitwise operators.

```
#include <stdio.h>


int isPowerOfTwo(int n) {
   return (n > 0) && ((n & (n - 1)) == 0);
}
int main() {
   int number;
   printf("Enter a number: ");
   scanf("%d", &number);


   if (isPowerOfTwo(number)) {
      printf("%d is a power of 2.\n", number);
   } else {
      printf("%d is not a power of 2.\n", number);
   }
   return 0;
}
```

Output:

```
Enter a number: 8
8 is a power of 2.
```

Date: 10/ 12 / 2024

9. Create a linked list or a dynamic array.

```
#include <stdio.h>

#include <stdlib.h>


// Define the structure for a linked list node

struct Node {

   int data;

   struct Node* next;

};


// Function to create a new node

struct Node* createNode(int data) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   if (!newNode) {

      printf("Memory allocation failed\n");

      exit(1);

   }

   newNode->data = data;

   newNode->next = NULL;

   return newNode;

}
```

Date: 10/ 12 / 2024

```
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = createNode(10);
    head->next = createNode(20);
    head->next->next = createNode(30);
    printf("Linked List: ");
    printList(head);
    return 0;
}
```

Output:

```
Linked List: 10 -> 20 -> 30 -> NULL
```

Date: 10/ 12 / 2024

10. Write a program to encode a decimal into binary and vice versa.

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

void decimal_to_binary(int decimal) {
    int binary[32], i = 0;

    while (decimal > 0) {
        binary[i] = decimal % 2;
        decimal /= 2;
        i++;
    }

    printf("Binary representation: ");
    for (int j = i - 1; j >= 0; j--) {
        printf("%d", binary[j]);
    }
    printf("\n");
}

int binary_to_decimal(char binary[]) {
    int decimal = 0, base = 1, len = strlen(binary);
    for (int i = len - 1; i >= 0; i--) {
        if (binary[i] == '1') {
            decimal += base;
```

Date: 10/ 12 / 2024

```
    }
        base *= 2;
    }
    return decimal;
}
int main() {
    int choice;
    printf("Choose: 1. Decimal to Binary 2. Binary to Decimal: ");
    scanf("%d", &choice);

    if (choice == 1) {
        int decimal;
        printf("Enter decimal number: ");
        scanf("%d", &decimal);
        decimal_to_binary(decimal);
    } else if (choice == 2) {
        char binary[32];
        printf("Enter binary number: ");
        scanf("%s", binary);
        printf("Decimal representation: %d\n", binary_to_decimal(binary));
    } else {
        printf("Invalid option.\n");
    }
    return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Choose: 1. Decimal to Binary 2. Binary to Decimal: 2

Enter binary number: 1100110

Decimal representation: 102
```

Date: 10/ 12 / 2024

11. Write a client-server chat application.(JAVA)

SERVER

```
import java.io.*;
import java.net.*;


public class ChatServer {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(12345)) {
            System.out.println("Server is running and waiting for a client...");
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected!");


            BufferedReader input = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter output = new PrintWriter(clientSocket.getOutputStream(), true);


            Thread readerThread = new Thread(() -> {
                try {
                    String clientMessage;
                    while ((clientMessage = input.readLine()) != null) {
                        System.out.println("Client: " + clientMessage);
                    }
                } catch (IOException e) {
                    System.out.println("Connection closed.");
                }
```

Date: 10/ 12 / 2024

```
});
        readerThread.start();


        BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));
        String serverMessage;
        while ((serverMessage = consoleInput.readLine()) != null) {
           output.println(serverMessage);
        }

      } catch (IOException e) {
        System.out.println("Server error: " + e.getMessage());
      }
   }
}
```

CLIENT

```
import java.io.*;
import java.net.*;


public class ChatClient {
   public static void main(String[] args) {
      try (Socket socket = new Socket("localhost", 12345)) {
        System.out.println("Connected to the server!")
```

Date: 10/ 12 / 2024

```
BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter output = new PrintWriter(socket.getOutputStream(), true);


        Thread readerThread = new Thread(() -> {
            try {
                String serverMessage;
                while ((serverMessage = input.readLine()) != null) {
                    System.out.println("Server: " + serverMessage);
                }
            } catch (IOException e) {
                System.out.println("Connection closed.");
            }
        });
        readerThread.start();


        BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));
        String clientMessage;
        while ((clientMessage = consoleInput.readLine()) != null) {
            output.println(clientMessage);
        }
    } catch (IOException e) {
        System.out.println("Client error: " + e.getMessage());
    }
  }
}
```

Date: 10/ 12 / 2024

Output:

Server

```
F:\semester 6\Information Network Security(INS)\lab\lab 1\chat>javac ChatServer.java ChatClient.java

F:\semester 6\Information Network Security(INS)\lab\lab 1\chat>java ChatServer

Server is running and waiting for a client...

Client connected!

hi

Client: hello

good morning

Client: how are you

i am fine
```

Client

```
F:\semester 6\Information Network Security(INS)\lab\lab 1\chat>java ChatClient

Connected to the server!

Server: hi

hello

Server: good morning

how are you

Server: i am fine
```

Date: 10/ 12 / 2024

13.Implement file transfer over a network.

SERVER

```
import java.io.*;
import java.net.*;


public class FileTransferServer {
   public static void main(String[] args) {
      final int PORT = 12345;


      try (ServerSocket serverSocket = new ServerSocket(PORT)) {
         System.out.println("Server is running and waiting for a connection...");
         Socket clientSocket = serverSocket.accept();
         System.out.println("Client connected!");


         // File to send
         File fileToSend = new File("file-to-send.txt");
         if (!fileToSend.exists()) {
            System.out.println("File does not exist.");
            return;
         }


         // Sending file
         try (BufferedInputStream fileInput = new BufferedInputStream(new FileInputStream(fileToSend));
              OutputStream outputStream = clientSocket.getOutputStream()) {
            System.out.println("Sending file: " + fileToSend.getName());
```

Date: 10/ 12 / 2024

```
byte[] buffer = new byte[4096];
            int bytesRead;
            while ((bytesRead = fileInput.read(buffer)) > 0) {
               outputStream.write(buffer, 0, bytesRead);
            }
            System.out.println("File sent successfully!");
         }
      } catch (IOException e) {
         System.out.println("Server error: " + e.getMessage());
      }
   }
}
```

Date: 10/ 12 / 2024

CLIENT

```
import java.io.*;

import java.net.*;

public class FileTransferClient {

   public static void main(String[] args) {

      final String SERVER_ADDRESS = "localhost";

      final int PORT = 12345;

      try (Socket socket = new Socket(SERVER_ADDRESS, PORT)) {

         System.out.println("Connected to the server!");

         File receivedFile = new File("received-file.txt");

         try (InputStream inputStream = socket.getInputStream();

              BufferedOutputStream fileOutput = new BufferedOutputStream(new
FileOutputStream(receivedFile))) {

            System.out.println("Receiving file...");

            byte[] buffer = new byte[4096];

            int bytesRead;

            while ((bytesRead = inputStream.read(buffer)) > 0) {

               fileOutput.write(buffer, 0, bytesRead);

            }

            System.out.println("File received successfully: " + receivedFile.getAbsolutePath());

         }

      } catch (IOException e) {

         System.out.println("Client error: " + e.getMessage());

      }

   }

}
```

Date: 10/ 12 / 2024

Output:

SERVER

```
F:\semester 6\Information Network Security(INS)\lab\lab 1\file>java FileTransferServer

Server is running and waiting for a connection...

Client connected!

Sending file: file-to-send.txt

File sent successfully!
```

CLIENT

```
F:\semester 6\Information Network Security(INS)\lab\lab 1\file>java FileTransferClient

Connected to the server!

Receiving file...

File received successfully: F:\semester 6\Information Network Security(INS)\lab\lab 1\file\received-file.txt
```

Date: 10/ 12 / 2024

Difference Between Active Attack and Passive Attack

Active Attack	**Passive Attack**
In an active attack, Modification in information takes place.	While in a passive attack, Modification in the information does not take place.
Active Attack is a danger to **Integrity** as well as **availability**.	Passive Attack is a danger to **Confidentiality**.
In an active attack, attention is on prevention.	While in passive attack attention is on detection.
Due to active attacks, the execution system is always damaged.	While due to passive attack, there is no harm to the system.
In an active attack, Victim gets informed about the attack.	While in a passive attack, Victim does not get informed about the attack.
In an active attack, System resources can be changed.	While in passive attack, System resources are not changing.
Active attack influences the services of the system.	While in a passive attack, information and messages in the system or network are acquired.
In an active attack, information collected through passive attacks is used during execution.	While passive attacks are performed by collecting information such as passwords, and messages by themselves.
An active attack is tough to restrict from entering systems or networks.	Passive Attack is easy to prohibit in comparison to active attack.
Can be easily detected.	Very difficult to detect.
The purpose of an active attack is to harm the ecosystem.	The purpose of a passive attack is to learn about the ecosystem.
In an active attack, the original information is modified.	In passive attack original information is Unaffected.
The duration of an active attack is short.	The duration of a passive attack is long.
The prevention possibility of active attack is High	The prevention possibility of passive attack is low.
Complexity is High	Complexity is low.

Date: 10/ 12 / 2024

Lab Practical #02:
Implement plain text to cipher text using operation of AND or XOR

1. XOR Code:

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

void xorEncryptDecrypt(const char text[], char output[], char key) {
    for (int i = 0; i < strlen(text); i++) {
        output[i] = text[i] ^ key;
    }
    output[strlen(text)] = '\0';
}

void printBinary(const char text[]) {
    for (int i = 0; i < strlen(text); i++) {
        for (int j = 7; j >= 0; j--) {
            printf("%d", (text[i] >> j) & 1);
        }
        printf(" ");
    }
    printf("\n");
}
```

Date: 10/ 12 / 2024

```
int main() {
    char text[100], encrypted[100], decrypted[100];
    char key;

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

    printf("Enter key (single character): ");
    scanf(" %c", &key);

    xorEncryptDecrypt(text, encrypted, key);
    printf("Encrypted text : %d \n", encrypted);

    printf("Encrypted text (as binary): ");
    printBinary(encrypted);

    xorEncryptDecrypt(encrypted, decrypted, key);
    printf("Decrypted text: %s\n", decrypted);
    return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter text: hello

Enter key (single character): k

Encrypted text : 6421975

Encrypted text (as binary): 00000011 00001110 00000111 00000111 00000100

Decrypted text: hello
```

2 AND Code:

```
#include <stdio.h>

#include <string.h>

void andEncrypt(const char text[], char output[], char key) {

   for (int i = 0; i < strlen(text); i++) {

      output[i] = text[i] & key;

   }

   output[strlen(text)] = '\0';

}

void printBinary(const char text[]) {

   for (int i = 0; i < strlen(text); i++) {

      for (int j = 7; j >= 0; j--) {

         printf("%d", (text[i] >> j) & 1);

      }

      printf(" ");

   }
```

Date: 10/ 12 / 2024

```
printf("\n");
}
int main() {
    char text[100], encrypted[100], decrypted[100];
    char key;

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

    printf("Enter key (single character): ");
    scanf(" %c", &key);

    andEncrypt(text, encrypted, key);

    printf("Encrypted text (as binary): ");
    printBinary(encrypted);

    return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter text: hello

Enter key (single character): k

Encrypted text (as binary): 01101000 01100001 01101000 01101000 01101011
```

Date: 10/ 12 / 2024

Lab Practical #03:
Implementation of Caesar Cipher Techniques.

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

void enc(char *str, int key) {
    int i;
    char str2[30];

    for (i = 0; i < strlen(str); i++) {
        if (str[i] >= 'a' && str[i] <= 'z') {
            str2[i] = ((str[i] - 'a' + key) % 26) + 'a';
        } else if (str[i] >= 'A' && str[i] <= 'Z') {
            str2[i] = ((str[i] - 'A' + key) % 26) + 'A';
        } else {
            str2[i] = str[i];
        }
    }
    str2[i] = '\0';
    strcpy(str, str2);
}
int main() {
    char str[30];
    int key;
    printf("Enter String for Encryption: ");
```

Date: 10/ 12 / 2024

```
  printf("Enter String for Encryption: ");
   fgets(str, sizeof(str), stdin);
   str[strcspn(str, "\n")] = 0;

   printf("Enter Key in integer: ");
   scanf("%d", &key);

   enc(str, key);

   printf("Encrypted String: %s\n", str);

   return 0;
}
```

Output:

```
Enter String for Encryption: welcome
Enter Key in integer: 4
Encrypted String: aipgsqi
```

Date: 10/ 12 / 2024

Lab Practical #04:
Implementation of monoalphabetic and polyalphabetic substitution cipher technique.

Monoalphabetic cipher:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void generateKey(char *inputKey, char *key) {
   int len = strlen(inputKey);
   int used[26] = {0};
   int index = 0;
   for (int i = 0; i < len; i++) {
      if (isalpha(inputKey[i])) {
         char c = toupper(inputKey[i]);
         if (!used[c - 'A']) {
            used[c - 'A'] = 1;
            key[index++] = c;
         }
      } }
   for (int i = 0; i < 26; i++) {
      if (!used[i]) {
         key[index++] = 'A' + i;
      }
   }
   key[26] = '\0'
```

Date: 10/ 12 / 2024

```
}
void encrypt(char *plaintext, char *substitution) {
    int len = strlen(plaintext);
    for (int i = 0; i < len; i++) {
        if (isupper(plaintext[i])) {
            int index = plaintext[i] - 'A';
            plaintext[i] = substitution[index];
        }
        else if (islower(plaintext[i])) {
            int index = plaintext[i] - 'a';
            plaintext[i] = tolower(substitution[index]);
        }
    }
}
void decrypt(char *ciphertext, char *substitution) {
    int len = strlen(ciphertext);
    char reverseSubstitution[26];
    for (int i = 0; i < 26; i++) {
        reverseSubstitution[substitution[i] - 'A'] = 'A' + i;
    }
    for (int i = 0; i < len; i++) {
        if (isupper(ciphertext[i])) {
            int index = ciphertext[i] - 'A';
            ciphertext[i] = reverseSubstitution[index];
        }
```

Date: 10/ 12 / 2024

```
else if (islower(ciphertext[i])) {
            int index = ciphertext[i] - 'a';
            ciphertext[i] = tolower(reverseSubstitution[index]);
        }
    }
}

int main() {
    char plaintext[100], ciphertext[100], inputKey[100], key[27];
    int choice;

    printf("Enter substitution key (any string): ");
    fgets(inputKey, sizeof(inputKey), stdin);

    inputKey[strcspn(inputKey, "\n")] = '\0';

    generateKey(inputKey, key);
    printf("Generated substitution key: %s\n", key);

    while (1) {
        printf("\nSelect an option:\n");
        printf("1. Encrypt\n");
        printf("2. Decrypt\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
```

Date: 10/ 12 / 2024

```
 scanf("%d", &choice);
      getchar();
      if (choice == 1) {
         printf("Enter plaintext: ");
         fgets(plaintext, sizeof(plaintext), stdin);
         plaintext[strcspn(plaintext, "\n")] = '\0';
         strcpy(ciphertext, plaintext);
         encrypt(ciphertext, key);
         printf("Encrypted text: %s\n", ciphertext);
      }
      else if (choice == 2) {
         printf("Enter ciphertext: ");
         fgets(ciphertext, sizeof(ciphertext), stdin);
         ciphertext[strcspn(ciphertext, "\n")] = '\0';
         decrypt(ciphertext, key);
         printf("Decrypted text: %s\n", ciphertext);
      }
      else if (choice == 3) {
         printf("Exiting...\n");
         break;
      } else {
         printf("Invalid choice, please try again.\n");
      }
   }
   return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter substitution key (any string): and rew cipher

Generated substitution key: ANDREWCIPHBFGJKLMOQSTUVXYZ


Select an option:

1. Encrypt

2. Decrypt

3. Exit

Enter your choice: 1

Enter plaintext: good morning

Encrypted text: ckkr gkojpjc


Select an option:

1. Encrypt

2. Decrypt

3. Exit

Enter your choice: 3

Exiting...
```

Date: 10/ 12 / 2024

Polyalphabetic cipher:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void encrypt(char *plaintext, char *key) {
   int len = strlen(plaintext);
   int keyLen = strlen(key);
   int j = 0;
   for (int i = 0; i < len; i++) {
      if (isalpha(plaintext[i])) {
         char shift = toupper(key[j % keyLen]) - 'A';
         if (isupper(plaintext[i])) {
            plaintext[i] = (plaintext[i] - 'A' + shift) % 26 + 'A';
         } else if (islower(plaintext[i])) {
            plaintext[i] = (plaintext[i] - 'a' + shift) % 26 + 'a';
         }
         j++;
      }
   }
}
void decrypt(char *ciphertext, char *key) {
   int len = strlen(ciphertext);
   int keyLen = strlen(key);
   int j = 0;
   for (int i = 0; i < len; i++) {
```

Date: 10/ 12 / 2024

```
 if (isalpha(ciphertext[i])) {
            char shift = toupper(key[j % keyLen]) - 'A';
            if (isupper(ciphertext[i])) {
                ciphertext[i] = (ciphertext[i] - 'A' - shift + 26) % 26 + 'A';
            } else if (islower(ciphertext[i])) {
                ciphertext[i] = (ciphertext[i] - 'a' - shift + 26) % 26 + 'a';
            }
            j++;
        }
    }
}
int main() {
    char plaintext[100], ciphertext[100], key[100];
    int choice;
    printf("Enter the key (any string): ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';
    while (1) {
        printf("\nSelect an option:\n");
        printf("1. Encrypt\n");
        printf("2. Decrypt\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();
```

Date: 10/ 12 / 2024

```
if (choice == 1) {
        printf("Enter plaintext: ");
        fgets(plaintext, sizeof(plaintext), stdin);
        plaintext[strcspn(plaintext, "\n")] = '\0';
        strcpy(ciphertext, plaintext);
        encrypt(ciphertext, key);
        printf("Encrypted text: %s\n", ciphertext);
    }
    else if (choice == 2) {
        printf("Enter ciphertext: ");
        fgets(ciphertext, sizeof(ciphertext), stdin);
        ciphertext[strcspn(ciphertext, "\n")] = '\0';
        decrypt(ciphertext, key);
        printf("Decrypted text: %s\n", ciphertext);
    }
    else if (choice == 3) {
        printf("Exiting...\n");
        break;
    } else {
        printf("Invalid choice, please try again.\n");
    }
  }
  return 0;
```

Date: 10/ 12 / 2024

Output:

```
Enter the key (any string): good


Select an option:
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter plaintext: grow more tree
Encrypted text: mfcz scfh zfsh


Select an option:
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 3
Exiting...
```

Date: 10/ 12 / 2024

Lab Practical #05:
Implementation of Playfair Cipher techniques..

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
void generateMatrix(char *key, char matrix[SIZE][SIZE]) {
   int used[26] = {0};
   int k = 0;
   for (int i = 0; i < strlen(key); i++) {
      char c = toupper(key[i]);
      if (isalpha(c) && c != 'J' && !used[c - 'A']) {
         matrix[k / SIZE][k % SIZE] = c;
         used[c - 'A'] = 1;
         k++;
      }

   for (char c = 'A'; c <= 'Z'; c++) {
      if (c != 'J' && !used[c - 'A']) {
         matrix[k / SIZE][k % SIZE] = c;
         used[c - 'A'] = 1;
         k++;
      }
   }
}
```

Date: 10/ 12 / 2024

```
void printMatrix(char matrix[SIZE][SIZE]) {
   printf("\nPlayfair Matrix:\n");
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         printf("%c ", matrix[i][j]);
      }
      printf("\n");
   }
}
void findPosition(char letter, char matrix[SIZE][SIZE], int *row, int *col) {
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         if (matrix[i][j] == letter) {
            *row = i;
            *col = j;
            return;
         }
      }
   }
}
void preparePlaintext(char *plaintext, char *preparedText) {
   int len = strlen(plaintext);
   int j = 0;
   for (int i = 0; i < len; i++) {
      if (isalpha(plaintext[i])) {
```

Date: 10/ 12 / 2024

```
            char c = toupper(plaintext[i]);
            if (i + 1 < len && toupper(plaintext[i]) == toupper(plaintext[i + 1])) {
                preparedText[j++] = c;
                preparedText[j++] = 'X';
                i++;
            } else {
                preparedText[j++] = c;
            }
        }
    }
    if (j % 2 != 0) {   preparedText[j++] = 'X';  }
    preparedText[j] = '\0';
}
void encryptPair(char a, char b, char matrix[SIZE][SIZE], char *encryptedPair) {
    int row1, col1, row2, col2;
    findPosition(a, matrix, &row1, &col1);
    findPosition(b, matrix, &row2, &col2);
    if (row1 == row2) {
        encryptedPair[0] = matrix[row1][(col1 + 1) % SIZE];
        encryptedPair[1] = matrix[row2][(col2 + 1) % SIZE];
    } else if (col1 == col2) {
        encryptedPair[0] = matrix[(row1 + 1) % SIZE][col1];
        encryptedPair[1] = matrix[(row2 + 1) % SIZE][col2];
    } else {
        encryptedPair[0] = matrix[row1][col2];
```

Date: 10/ 12 / 2024

```
        encryptedPair[1] = matrix[row2][col1];
    }
}
void encrypt(char *plaintext, char *key, char *ciphertext) {
    char matrix[SIZE][SIZE];
    char preparedText[100];
    generateMatrix(key, matrix);
    printMatrix(matrix);
    preparePlaintext(plaintext, preparedText);

    int j = 0;
    for (int i = 0; i < strlen(preparedText); i += 2) {
        char encryptedPair[3];
        encryptPair(preparedText[i], preparedText[i + 1], matrix, encryptedPair);
        ciphertext[j++] = encryptedPair[0];
        ciphertext[j++] = encryptedPair[1];
    }
    ciphertext[j] = '\0';
}
void decryptPair(char a, char b, char matrix[SIZE][SIZE], char *decryptedPair) {
    int row1, col1, row2, col2;
    findPosition(a, matrix, &row1, &col1);
    findPosition(b, matrix, &row2, &col2);
    if (row1 == row2) {
        decryptedPair[0] = matrix[row1][(col1 - 1 + SIZE) % SIZE];
        decryptedPair[1] = matrix[row2][(col2 - 1 + SIZE) % SIZE];
```

Date: 10/ 12 / 2024

```
} else if (col1 == col2) {
        decryptedPair[0] = matrix[(row1 - 1 + SIZE) % SIZE][col1];
        decryptedPair[1] = matrix[(row2 - 1 + SIZE) % SIZE][col2];
    } else {
        decryptedPair[0] = matrix[row1][col2];
        decryptedPair[1] = matrix[row2][col1];
    }
}
void decrypt(char *ciphertext, char *key, char *plaintext) {
    char matrix[SIZE][SIZE];
    generateMatrix(key, matrix);
    printMatrix(matrix);

    int j = 0;
    for (int i = 0; i < strlen(ciphertext); i += 2) {
        char decryptedPair[3];
        decryptPair(ciphertext[i], ciphertext[i + 1], matrix, decryptedPair);
        plaintext[j++] = decryptedPair[0];
        plaintext[j++] = decryptedPair[1];
    }
    plaintext[j] = '\0';
}
int main() {
    char plaintext[100], ciphertext[100], key[100];
    int choice;
```

Date: 10/ 12 / 2024

```
printf("Enter the key (any string): ");
fgets(key, sizeof(key), stdin);
key[strcspn(key, "\n")] = '\0';
while (1) {
    printf("\nSelect an option:\n");
    printf("1. Encrypt\n");
    printf("2. Decrypt\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar();
    if (choice == 1) {
        printf("Enter plaintext: ");
        fgets(plaintext, sizeof(plaintext), stdin);
        plaintext[strcspn(plaintext, "\n")] = '\0';
        encrypt(plaintext, key, ciphertext);
        printf("Encrypted text: %s\n", ciphertext);
    }
    else if (choice == 2) {
        printf("Enter ciphertext: ");
        fgets(ciphertext, sizeof(ciphertext), stdin);
        ciphertext[strcspn(ciphertext, "\n")] = '\0';
        decrypt(ciphertext, key, plaintext);
        printf("Decrypted text: %s\n", plaintext);
    }
```

Date: 10/ 12 / 2024

```
else if (choice == 3) {
        printf("Exiting...\n");
        break;
    } else {
        printf("Invalid choice, please try again.\n");
    }
  }

  return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter the key (any string): keyword

Select an option:
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter plaintext: why donot you

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

Select an option:
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter ciphertext: YIEAESKZWKVZ
```

Date: 10/ 12 / 2024

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

Select an option:
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 3
Exiting...
```

Date: 10/ 12 / 2024

Lab Practical #06:
Implementation of Hill Cipher techniques.

HILL Cipher 2x2 and 3x3 technique

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
#define SIZEE 3
int matrixSize = 2;


void getKeyMatrix(char key[], int keyMatrix[2][2])
{
   int k = 0;
   for (int i = 0; i < matrixSize; i++)
   {
      for (int j = 0; j < matrixSize; j++)
      {
         keyMatrix[i][j] = key[k] - 'A';
         k++;
      }
   }
}
```

Date: 10/ 12 / 2024

```
void encrypt(char plaintext[], char key[], char ciphertext[])
{
   int keyMatrix[2][2];
   getKeyMatrix(key, keyMatrix);
   int plaintextVector[2];
   int resultVector[2];
   int i;
   for (i = 0; i < strlen(plaintext); i += 2)
   {
      if (i + 1 >= strlen(plaintext))
      {
         plaintext[i + 1] = 'X';
      }
      for (int j = 0; j < 2; j++)
      {
         plaintextVector[j] = plaintext[i + j] - 'A';
      }
      for (int j = 0; j < 2; j++)
      {
         resultVector[j] = 0;
         for (int k = 0; k < 2; k++)
         {
            resultVector[j] += keyMatrix[j][k] * plaintextVector[k];
         }
         resultVector[j] %= 26;
      }
```

Date: 10/ 12 / 2024

```
    for (int j = 0; j < 2; j++) {  ciphertext[i + j] = resultVector[j] + 'A';   }
    }
    ciphertext[i] = '\0';
}
void findInverseKeyMatrix(int keyMatrix[2][2], int inverseKeyMatrix[2][2])
{
    int determinant = (keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] * keyMatrix[1][0]) % 26;
    if (determinant < 0)
        determinant += 26;
    int inverseDeterminant = -1;
    for (int i = 0; i < 26; i++)
    {
        if ((determinant * i) % 26 == 1)
        {
            inverseDeterminant = i;
            break;
        }
    }
    if (inverseDeterminant == -1)
    {
        printf("Key matrix is not invertible.\n");
        exit(1);
    }
    inverseKeyMatrix[0][0] = (keyMatrix[1][1] * inverseDeterminant) % 26;
    inverseKeyMatrix[0][1] = (-keyMatrix[0][1] * inverseDeterminant) % 26;
```

Date: 10/ 12 / 2024

```
    inverseKeyMatrix[1][0] = (-keyMatrix[1][0] * inverseDeterminant) % 26;
    inverseKeyMatrix[1][1] = (keyMatrix[0][0] * inverseDeterminant) % 26;
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            if (inverseKeyMatrix[i][j] < 0)   {    inverseKeyMatrix[i][j] += 26;  }
        }
    }
}
void decrypt(char ciphertext[], char key[], char plaintext[])
{
    int keyMatrix[2][2], inverseKeyMatrix[2][2];
    getKeyMatrix(key, keyMatrix);
    findInverseKeyMatrix(keyMatrix, inverseKeyMatrix);
    int ciphertextVector[2];
    int resultVector[2];
    for (int i = 0; i < strlen(ciphertext); i += 2)
    {
        for (int j = 0; j < 2; j++)
        {     ciphertextVector[j] = ciphertext[i + j] - 'A';      }
        for (int j = 0; j < 2; j++)
        {
            resultVector[j] = 0;
            for (int k = 0; k < 2; k++)
            {
```

Date: 10/ 12 / 2024

```
    resultVector[j] += inverseKeyMatrix[j][k] * ciphertextVector[k];
            }
            resultVector[j] %= 26;
        }
        for (int j = 0; j < 2; j++)
        {
            plaintext[i + j] = resultVector[j] + 'A';
        }
    }
    plaintext[strlen(ciphertext)] = '\0';
}
int determinant(int matrixaa[SIZEE][SIZEE])
{
    return (matrixaa[0][0] * (matrixaa[1][1] * matrixaa[2][2] - matrixaa[1][2] * matrixaa[2][1]) -
            matrixaa[0][1] * (matrixaa[1][0] * matrixaa[2][2] - matrixaa[1][2] * matrixaa[2][0]) +
            matrixaa[0][2] * (matrixaa[1][0] * matrixaa[2][1] - matrixaa[1][1] * matrixaa[2][0]));
}
int modInverse(int a, int m)
{
    a = a % m;
    for (int x = 1; x < m; x++)
    {
        if ((a * x) % m == 1)
        {
            return x;
        }
```

Date: 10/ 12 / 2024

```
    }
    return -1;
}
void hillCipher(int key[SIZEE][SIZEE], char *input)
{
    int matrixaa[SIZEE][SIZEE];
    int resultaa[SIZEE];
    int length = strlen(input);
    char output[length + 1];
    for (int i = 0; i < length; i += SIZEE)
    {
        for (int j = 0; j < SIZEE; j++)
        {
            if (i + j < length) { matrixaa[j][0] = input[i + j] - 'A'; }
            else { matrixaa[j][0] = 'X' - 'A'; }
        }
        for (int j = 0; j < SIZEE; j++)
        {
            resultaa[j] = 0;
            for (int k = 0; k < SIZEE; k++)
            {
                resultaa[j] += key[j][k] * matrixaa[k][0];
            }
            output[i / SIZEE * SIZEE + j] = (resultaa[j] % 26) + 'A';
        }
    }
```

Date: 10/ 12 / 2024

```
    output[length] = '\0';
    printf("Encrypted Output: %s\n", output);
}
void initializeKeyMatrix(int ekey[SIZEE][SIZEE], char input[100]) {
    int count = 0;
    for (int i = 0; i < SIZEE; i++) {
        for (int j = 0; j < SIZEE; j++) {
            if (count < strlen(input) && input[count] >= 'a' && input[count] <= 'z') {
                ekey[i][j] = input[count] - 'a';
            } else {   ekey[i][j] = 23;  }
            count++;
        }
    }
     for (int i = 0; i < SIZEE; i++) {
        for (int j = 0; j < SIZEE; j++) {
            if (ekey[i][j] == 0 && (i != 0 || j != 0)) {
                ekey[i][j] = 23;
            }
        }
    }
    printf("The 3x3 key matrix is:\n");
    for (int i = 0; i < SIZEE; i++) {
        for (int j = 0; j < SIZEE; j++) {
            printf("%d ", ekey[i][j]);
        }
```

Date: 10/ 12 / 2024

```
        printf("\n");
    }
}
int determinanta(int matrix[3][3]) {
    return matrix[0][0] * (matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1])
         - matrix[0][1] * (matrix[1][0] * matrix[2][2] - matrix[1][2] * matrix[2][0])
         + matrix[0][2] * (matrix[1][0] * matrix[2][1] - matrix[1][1] * matrix[2][0]);
}
int mod_inversea(int num, int mod) {
    num = num % mod;
    for (int x = 1; x < mod; x++) {
        if ((num * x) % mod == 1)
            return x;
    }
    return -1;
}
void adjugate_matrix(int matrix[3][3], int adj[3][3]) {
    adj[0][0] =  matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1];
    adj[0][1] = -(matrix[0][1] * matrix[2][2] - matrix[0][2] * matrix[2][1]);
    adj[0][2] =  matrix[0][1] * matrix[1][2] - matrix[0][2] * matrix[1][1];
    adj[1][0] = -(matrix[1][0] * matrix[2][2] - matrix[1][2] * matrix[2][0]);
    adj[1][1] =  matrix[0][0] * matrix[2][2] - matrix[0][2] * matrix[2][0];
    adj[1][2] = -(matrix[0][0] * matrix[1][2] - matrix[0][2] * matrix[1][0]);
    adj[2][0] =  matrix[1][0] * matrix[2][1] - matrix[1][1] * matrix[2][0];
    adj[2][1] = -(matrix[0][0] * matrix[2][1] - matrix[0][1] * matrix[2][0]);
    adj[2][2] =  matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
```

Date: 10/ 12 / 2024

```
}
void inverse_matrix(int matrix[3][3], int inverse[3][3], int mod) {
    int det = determinanta(matrix);
    det = det % mod;
    if (det < 0) det += mod;
    int det_inv = mod_inversea(det, mod);
    if (det_inv == -1) {
        printf("Key matrix is not invertible modulo %d.\n", mod);
        exit(1);
    }
    int adj[3][3];
    adjugate_matrix(matrix, adj);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            inverse[i][j] = (adj[i][j] * det_inv) % mod;
            if (inverse[i][j] < 0) inverse[i][j] += mod;
        }
    }
}
void decryptaaa(char* ciphertext, int key[3][3]) {
    int key_inverse[3][3];
    int mod = 26;
    int len = strlen(ciphertext);
    while (len % 3 != 0) {
        strcat(ciphertext, "X");
        len++;
```

Date: 10/ 12 / 2024

```
 }

  inverse_matrix(key, key_inverse, mod);

  printf("Decrypted text: ");

  for (int i = 0; i < len; i += 3) {

    int c[3] = {ciphertext[i] - 'A', ciphertext[i + 1] - 'A', ciphertext[i + 2] - 'A'};

    int p[3] = {0};

    for (int row = 0; row < 3; row++) {

      for (int col = 0; col < 3; col++) {

        p[row] += key_inverse[row][col] * c[col];

      }

      p[row] = p[row] % mod;

      if (p[row] < 0) p[row] += mod;

    }

    printf("%c%c%c", p[0] + 'A', p[1] + 'A', p[2] + 'A');

  }

  printf("\n");

}


int main()

{

  int keySize;

  char plaintext[MAX], ciphertext[MAX], key[5];

  int choice;

  int ekey[SIZEE][SIZEE];

  char enc_pt[100];

  char cipher_text[100];
```

Date: 10/ 12 / 2024

```
int key_matrix[SIZEE][SIZEE];
   char k[100],w[100];
   while (1)
   {
      printf("\n--- Hill Cipher ---\n");
      printf("Choose key size:\n");
      printf("2: 2x2 key matrix\n");
      printf("3: 3x3 key matrix\n");
      printf("1: exit\n");
      scanf("%d", &keySize);
      if (keySize == 2)
      {
         printf("2x2\n1. Encrypt\n");
         printf("2. Decrypt\n");
         printf("3. Exit\n");
         printf("Enter your choice: ");
         scanf("%d", &choice);
         switch (choice)
         {
         case 1:
            printf("Enter plaintext (in uppercase, multiple of 2 length): ");
            scanf("%s", plaintext);
            printf("Enter 4-character key (in uppercase): ");
            scanf("%s", key);
            encrypt(plaintext, key, ciphertext);
            printf("Ciphertext: %s\n", ciphertext);
```

Date: 10/ 12 / 2024

```
break;
        case 2:
           printf("Enter ciphertext (in uppercase, multiple of 2 length): ");
           scanf("%s", ciphertext);
           printf("Enter 4-character key (in uppercase): ");
           scanf("%s", key);
           decrypt(ciphertext, key, plaintext);
           printf("Plaintext: %s\n", plaintext);
           break;
        case 3:
           printf("Exiting...\n");
           exit(0);
        default:
           printf("Invalid choice. Try again.\n");
        }
     }
     else if (keySize == 3)
     {
        printf("3x3 \n1. Encrypt\n");
        printf("2. Decrypt\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
```

Date: 10/ 12 / 2024

```
printf("Enter the  Plaintext:( A-Z): ");
            scanf("%s", enc_pt);
            printf("Enter the 3x3 key matrix (9 characters a-z in): ");
            scanf("%s", k);
            initializeKeyMatrix(ekey, k);
            hillCipher(ekey, enc_pt);
            break;
        case 2:
            printf("Enter the cipher text:( A-Z): ");
            scanf("%s", cipher_text);
            printf("Enter the 3x3 key matrix (9  characters a-z in): ");
            scanf("%s", w);
            initializeKeyMatrix(key_matrix, w);
            printf("Decrypted Text: ");
            decryptaaa(cipher_text, key_matrix);
            break;
        case 3:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice. Try again.\n");
        }
    } else if (keySize == 1)
    { printf("Exiting...\n");
        exit(0);
    }
```

Date: 10/ 12 / 2024

```
else
    {
        printf("Invalid choice. Try again.\n");
    }
  }
  return 0;
}
```

Output:

```
--- Hill Cipher ---
Choose key size:
2: 2x2 key matrix
3: 3x3 key matrix
1: exit
2
2x2
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter plaintext (in uppercase, multiple of 2 length): EXAM
Enter 4-character key (in uppercase): HILL
Ciphertext: ELSC
```

Date: 10/ 12 / 2024

```
--- Hill Cipher ---

Choose key size:

2: 2x2 key matrix

3: 3x3 key matrix

1: exit

2

2x2

1. Encrypt

2. Decrypt

3. Exit

Enter your choice: 2

Enter ciphertext (in uppercase, multiple of 2 length): ELSC

Enter 4-character key (in uppercase): HILL

Plaintext: EXAM

--- Hill Cipher ---

Choose key size:

2: 2x2 key matrix

3: 3x3 key matrix

1: exit

3

3x3

1. Encrypt

2. Decrypt

3. Exit

Enter your choice: 1

Enter the  Plaintext:( A-Z): ATTACKISTONIGHT
```

Date: 10/ 12 / 2024

```
Enter the 3x3 key matrix (9 characters a-z in): dkuujrjer

The 3x3 key matrix is:

3 10 20

20 9 17

9 4 17

Encrypted Output: YAJMGWMVZUNCAMP

--- Hill Cipher ---

Choose key size:

2: 2x2 key matrix

3: 3x3 key matrix

1: exit

3

3x3

1. Encrypt

2. Decrypt

3. Exit

Enter your choice: 2

Enter the cipher text:( A-Z): YAJMGWMVZUNCAMP

Enter the 3x3 key matrix (9  characters a-z in): dkuujrjer

The 3x3 key matrix is:

3 10 20

20 9 17

9 4 17

Decrypted Text: Decrypted text: ATTACKISTONIGHT

--- Hill Cipher ---

Choose key size:
```

Date: 10/ 12 / 2024

```
2: 2x2 key matrix

3: 3x3 key matrix

1: exit

3

3x3

1. Encrypt

2. Decrypt

3. Exit

Enter your choice: 3

Exiting...


------------------------------
```

Date: 10/ 12 / 2024

Lab Practical #07:

1. **Implementation of rail fence transposition techniques.**
2. **Implementation of row/columns transposition techniques**

Rail fence transposition techniques

```
#include <stdio.h>

#include <string.h>

void railFenceEncrypt(char* text, int rails) {

   int len = strlen(text);

   char rail[rails][len];

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

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

         rail[i][j] = ' ';

      }

   }

   int row = 0, col = 0;

   int dir_down = 0;

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

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

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

         dir_down = !dir_down;

      }

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

   }

    printf("Encrypted text: ");

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

Date: 10/ 12 / 2024

```
  for (int j = 0; j < len; j++) {
          if (rail[i][j] != ' ') {
             printf("%c", rail[i][j]);
          }
       }
    }
    printf("\n");
}
int main() {
    char text[100];
    int rails;
    printf("Enter the text to encrypt: ");
    fgets(text, sizeof(text), stdin);
    text[strcspn(text, "\n")] = '\0';
    printf("Enter the number of rails: ");
    scanf("%d", &rails);
    if (rails < 2) {
        printf("Number of rails must be greater than 1.\n");
        return 1;
    }
    printf("Original text: %s\n", text);
    railFenceEncrypt(text, rails);

    return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter the text to encrypt: darshanuniversity

Enter the number of rails: 3

Original text: darshanuniversity

Encrypted text: dhnryasauiestrnvi



Enter the text to encrypt: niravkagathara

Enter the number of rails: 2

Original text: niravkagathara

Encrypted text: nrvaahriakgtaa
```

Date: 10/ 12 / 2024

Row/columns transposition techniques

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_LEN 1000
char* encrypt(int rows, int cols, int msg_len, const char* msg, int col_order[]) {
    int index = 0;
    char matrix[rows][cols];
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            if (index >= msg_len) {
                matrix[r][c] = '_';
            } else {
                matrix[r][c] = msg[index++];
            }
        }
    }
    static char cipher[MAX_LEN];
    cipher[0] = '\0';
    for (int c = 0; c < cols; c++) {
        int col_pos = col_order[c] - 1;
        for (int r = 0; r < rows; r++) {
            char str[2] = {matrix[r][col_pos], '\0'};
            strcat(cipher, str);
        }
    }
```

Date: 10/ 12 / 2024

```
    return cipher;
}
char* decrypt(int rows, int cols, const char* cipher, int col_order[]) {
    char matrix[rows][cols];
    int index = 0;
    for (int c = 0; c < cols; c++) {
        int col_pos = col_order[c] - 1;
        for (int r = 0; r < rows; r++) {
            matrix[r][col_pos] = cipher[index++];
        }
    }
    static char message[MAX_LEN];
    message[0] = '\0';
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            if (matrix[r][c] == '_') {
                matrix[r][c] = ' ';
            }
            char str[2] = {matrix[r][c], '\0'};
            strcat(message, str);
        }
    }
    return message;
}
int main() {
    char msg[MAX_LEN];
```

Date: 10/ 12 / 2024

```
char key[MAX_LEN];
    printf("Enter the message to encrypt: ");
    fgets(msg, sizeof(msg), stdin);
    msg[strcspn(msg, "\n")] = '\0';
    printf("Enter the key as space-separated numbers (e.g., '6 3 4 2 5 1'): ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';
    int cols = 0;
    int col_order[100];
    char *token = strtok(key, " ");
    while (token != NULL) {
        col_order[cols++] = atoi(token);
        token = strtok(NULL, " ");
    }
    int sorted_order[100];
    for (int i = 0; i < cols; i++) {
        sorted_order[i] = col_order[i];
    }
    for (int i = 0; i < cols; i++) {
        for (int j = i + 1; j < cols; j++) {
            if (sorted_order[i] > sorted_order[j]) {
                int temp = sorted_order[i];
                sorted_order[i] = sorted_order[j];
                sorted_order[j] = temp;
            }
        }
```

Date: 10/ 12 / 2024

```
}
    int final_order[100];
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < cols; j++) {
            if (col_order[j] == sorted_order[i]) {
                final_order[i] = j + 1;
            }
        }
    }
    int msg_len = strlen(msg);

    int rows = msg_len / cols;
    if (msg_len % cols != 0) {
        rows += 1;
    }

    char* encrypted_msg = encrypt(rows, cols, msg_len, msg, final_order);
    printf("Encrypted Message: %s\n", encrypted_msg);

    char* decrypted_msg = decrypt(rows, cols, encrypted_msg, final_order);
    printf("Decrypted Message: %s\n", decrypted_msg);
    return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter the message to encrypt: ATTACKPOSTPONDINVADODARA

Enter the key as space-separated numbers (e.g., '6 3 4 2 5 1'): 4 2 3 5

Encrypted Message: TKTDAATPPIDRACSNVDAOONOA

Decrypted Message: ATTACKPOSTPONDINVADODARA
```

Date: 10/ 12 / 2024

Lab Practical #08:

Implementation of Block Cipher techniques.

Electronic Code Book(ECB):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define BLOCK_SIZE 8
void encrypt_block(char *input, char *key, char *output) {
    for (int i = 0; i < BLOCK_SIZE; i++) {
        output[i] = input[i] ^ key[i % strlen(key)];
    }
}
void decrypt_block(char *input, char *key, char *output) {
    for (int i = 0; i < BLOCK_SIZE; i++) {
        output[i] = input[i] ^ key[i % strlen(key)];
    }
}
void pad_input(char *input, int *length) {
    int padding = BLOCK_SIZE - (*length % BLOCK_SIZE);
    for (int i = 0; i < padding; i++) {
        input[*length + i] = (char)padding;
    }
    *length += padding;
}
void remove_padding(char *input, int *length) {
    int padding = input[*length - 1];
    *length -= padding;
```

Date: 10/ 12 / 2024

```
    input[*length] = '\0';
}
int main() {
    char input[128], key[16], encrypted[128], decrypted[128];
    int input_len;
    printf("Enter plaintext (max 128 characters): ");
    fgets(input, sizeof(input), stdin);

    input_len = strlen(input);
    if (input[input_len - 1] == '\n') input[--input_len] = '\0';

    printf("Enter encryption key (max 16 characters): ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';
    pad_input(input, &input_len);
    for (int i = 0; i < input_len; i += BLOCK_SIZE) {
        encrypt_block(input + i, key, encrypted + i);
    }

    printf("Encrypted text: ");
    for (int i = 0; i < input_len; i++) {
        printf("%02X", (unsigned char)encrypted[i]);
    }
    printf("\n");
    for (int i = 0; i < input_len; i += BLOCK_SIZE) {
        decrypt_block(encrypted + i, key, decrypted + i);
```

Date: 10/ 12 / 2024

```
  }

   remove_padding(decrypted, &input_len);

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

   return 0;

}
```

Output:

```
Input: HelloWorld!

Key: secret123

Output:

Encrypted text: E6B5C49E8A6F7C3A...

Decrypted text: HelloWorld!
```

Date: 10/ 12 / 2024

Lab Practical #09:

Implementation of RSA .

```
#include <stdio.h>
#include <stdlib.h>
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
int findPrivateKey(int e, int phi_n) {
    int j = 1;
    while (1) {
        if ((1 + j * phi_n) % e == 0) {
            return (1 + j * phi_n) / e;
        }  j++;
    }}
int findRelativePrimeE(int phi_n) {
    int e = 2; // Start checking from 2
    while (gcd(e, phi_n) != 1) {
        e++; // Increment until we find a valid `e`
    }
    return e;
}
```

Date: 10/ 12 / 2024

```
long long modExp(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}
int main() {
    int p, q, e, choice;
    printf("Enter first prime number (p): ");
    scanf("%d", &p);
    printf("Enter second prime number (q): ");
    scanf("%d", &q);
    int n = p * q;
    int phi_n = (p - 1) * (q - 1);
    printf("\nChoose an option for public exponent e:\n");
    printf("1. Enter your own value of e\n");
    printf("2. Use an automatically selected e (relative prime to f(n))\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);
```

Date: 10/ 12 / 2024

```
if (choice == 1) {
      printf("Enter a value for e (must be coprime with f(n) = %d): ", phi_n);
      scanf("%d", &e);
      if (gcd(e, phi_n) != 1) {
         printf("Invalid choice! e is not coprime with f(n). Exiting.\n");
         return 1;
      }
   } else {
      e = findRelativePrimeE(phi_n);
      printf("Automatically chosen e: %d\n", e);
   }
   int d = findPrivateKey(e, phi_n);

   printf("\nPublic Key: (e = %d, n = %d)\n", e, n);
   printf("Private Key: (d = %d, n = %d)\n", d, n);
   int plaintext;

   printf("\nEnter the message (as a number) to encrypt: ");
   scanf("%d", &plaintext);

   long long ciphertext = modExp(plaintext, e, n);
   printf("Encrypted Message: %lld\n", ciphertext);
   long long decryptedMessage = modExp(ciphertext, d, n);
   printf("Decrypted Message: %lld\n", decryptedMessage);
   return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter first prime number (p): 7

Enter second prime number (q): 11


Choose an option for public exponent e:

1. Enter your own value of e

2. Use an automatically selected e (relative prime to f(n))

Enter your choice (1 or 2): 1

Enter a value for e (must be coprime with f(n) = 60): 7


Public Key: (e = 7, n = 77)

Private Key: (d = 43, n = 77)


Enter the message (as a number) to encrypt: 6

Encrypted Message: 41

Decrypted Message: 6
```

Date: 10/ 12 / 2024

Lab Practical #10:

Implementation of Diffie hellman key exchange techniques.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int is_primitive_root(int alpha, int q) {
    int seen[q];
    for (int i = 0; i < q; i++) seen[i] = 0;
    int value = 1;
    for (int i = 0; i < q - 1; i++) {
        value = (value * alpha) % q;
        if (seen[value] == 1) return 0;
        seen[value] = 1;
    }
    return 1;
}
int find_primitive_root(int q) {
    for (int alpha = 2; alpha < q; alpha++) {
        if (is_primitive_root(alpha, q)) return alpha;
    }
    return -1; // No primitive root found (should not happen for prime q)
}
long long power_mod(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) result = (result * base) % mod;
        base = (base * base) % mod;
```

Date: 10/ 12 / 2024

```
        base = (base * base) % mod;
        exp /= 2;
    } return result;
}
int main() {
    int q, alpha, choice;
    long long Xa, Xb, Ya, Yb, Ka, Kb;
    printf("Enter a prime number (q): ");
    scanf("%d", &q);
    printf("Choose:\n1. Enter a primitive root manually\n2. Auto-generate a primitive root\n");
    scanf("%d", &choice);
    if (choice == 1) {
        printf("Enter a primitive root (alpha) of %d: ", q);
        scanf("%d", &alpha);
        if (!is_primitive_root(alpha, q)) {
            printf("Error: %d is not a primitive root of %d. Exiting...\n", alpha, q);
            return 1;
        }
    } else {
        alpha = find_primitive_root(q);
        if (alpha == -1) {
            printf("Error: Could not find a primitive root for %d. Exiting...\n", q);
            return 1;
        }
        printf("Auto-generated primitive root (alpha): %d\n", alpha);
    }
```

Date: 10/ 12 / 2024

```
    printf("Enter private key for User A (Xa): ");
    scanf("%lld", &Xa);
    printf("Enter private key for User B (Xb): ");
    scanf("%lld", &Xb);
    Ya = power_mod(alpha, Xa, q);
    Yb = power_mod(alpha, Xb, q);
    Ka = power_mod(Yb, Xa, q); // Ka = Yb^Xa mod q
    Kb = power_mod(Ya, Xb, q); // Kb = Ya^Xb mod q
    printf("\nPublic Key for User A (Ya = alpha^Xa mod q): %lld\n", Ya);
    printf("Public Key for User B (Yb = alpha^Xb mod q): %lld\n", Yb);
    printf("Shared Secret Key (Ka = Yb^Xa mod q): %lld\n", Ka);
    printf("Shared Secret Key (Kb = Ya^Xb mod q): %lld\n", Kb);
    if (Ka == Kb) {
        printf("\nVerification successful! Both users have the same shared secret key: %lld\n", Ka);
    } else {
        printf("\nError: Shared keys do not match! Key exchange failed.\n");
    }
    return 0;
}
```

Date: 10/ 12 / 2024

Output:

```
Enter a prime number (q): 23

Choose:

1. Enter a primitive root manually

2. Auto-generate a primitive root

1

Enter a primitive root (alpha) of 23: 5

Enter private key for User A (Xa): 6

Enter private key for User B (Xb): 15


Public Key for User A (Ya = alpha^Xa mod q): 8

Public Key for User B (Yb = alpha^Xb mod q): 19

Shared Secret Key (Ka = Yb^Xa mod q): 2

Shared Secret Key (Kb = Ya^Xb mod q): 2


Verification successful! Both users have the same shared secret key: 2
```

Date: 10/ 12 / 2024

Lab Practical #09:

Implementation of AES and DES Algorithm.

Program:

DES encryption and decryption:

```
#include <stdio.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

void initialize_permutation(int **table, int size, const int values[]) {

    *table = (int *)malloc(size * sizeof(int));

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

        (*table)[i] = values[i];

    }}

uint64_t permute(uint64_t block, int *table, int size) {

    uint64_t result = 0;

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

        result |= ((block >> (64 - table[i])) & 1) << (size - i - 1);

    }

    return result;

}

uint64_t des_encrypt(uint64_t plaintext, uint64_t key, int *IP, int *IP_INV,
int *PC1, int *PC2, int *E, int *P) {

    plaintext = permute(plaintext, IP, 64);

    printf("\nAfter Initial Permutation: %llx", plaintext);

    uint32_t L = (plaintext >> 32) & 0xFFFFFFFF;

    uint32_t R = plaintext & 0xFFFFFFFF;

    printf("\nInitial L: %x, R: %x", L, R);
```

Date: 10/ 12 / 2024

```
 for (int i = 0; i < 16; i++) {
        uint32_t old_R = R;
        uint64_t expanded_R = permute(R, E, 48);
        expanded_R ^= key;
        uint32_t substituted = expanded_R & 0xFFFFFFFF;
        uint32_t permuted = permute(substituted, P, 32);
        R = L ^ permuted;
        L = old_R;
        printf("\nRound %d -> L: %x, R: %x", i + 1, L, R);
    }
uint64_t pre_output = ((uint64_t)R << 32) | L;
    uint64_t ciphertext = permute(pre_output, IP_INV, 64);
    return ciphertext;
}
int main() {
    uint64_t plaintext, key;
    printf("Enter 64-bit plaintext (in hex): ");
    scanf("%llx", &plaintext);
    printf("Enter 64-bit key (in hex): ");
    scanf("%llx", &key);
  int *IP, *IP_INV, *PC1, *PC2, *E, *P;
    int ip_values[] = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
                        62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
                        57, 49, 41, 33, 25, 17,  9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
                        61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 };
    initialize_permutation(&IP, 64, ip_values);
```

Date: 10/ 12 / 2024

```
int ip_inv_values[] = { 40,  8, 48, 16, 56, 24, 64, 32, 39,  7, 47, 15, 55, 23, 63, 31,
                        38,  6, 46, 14, 54, 22, 62, 30, 37,  5, 45, 13, 53, 21, 61, 29,
                        36,  4, 44, 12, 52, 20, 60, 28, 35,  3, 43, 11, 51, 19, 59, 27,
                        34,  2, 42, 10, 50, 18, 58, 26, 33,  1, 41,  9, 49, 17, 57, 25 };
   initialize_permutation(&IP_INV, 64, ip_inv_values);
   int pc1_values[] = { 57, 49, 41, 33, 25, 17,  9, 1, 58, 50, 42, 34, 26, 18,
                        10,  2, 59, 51, 43, 35, 27, 19, 11,  3, 60, 52, 44, 36,
                        63, 55, 47, 39, 31, 23, 15,  7, 62, 54, 46, 38, 30, 22,
                        14,  6, 61, 53, 45, 37, 29, 21, 13,  5, 28, 20, 12,  4 };
   initialize_permutation(&PC1, 56, pc1_values);
   int e_values[] = { 32,  1,  2,  3,  4,  5,  4,  5,  6,  7,  8,  9,
                       8,  9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,
                      16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
                      24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32,  1 };
   initialize_permutation(&E, 48, e_values);
   int p_values[] = { 16,  7, 20, 21, 29, 12, 28, 17,  1, 15, 23, 26,
                       5, 18, 31, 10,  2,  8, 24, 14, 32, 27,  3,  9,
                      19, 13, 30,  6, 22, 11,  4, 25 };
   initialize_permutation(&P, 32, p_values);
   uint64_t ciphertext = des_encrypt(plaintext, key, IP, IP_INV, PC1, PC2, E, P);
   printf("\nCiphertext: %llx\n", ciphertext);
   free(IP); free(IP_INV); free(PC1); free(PC2); free(E); free(P);
   return 0;
}
```

Date: 10/ 12 / 2024

Output:

Encryption:

Enter 64-bit plaintext (in hex): abc123

Enter 64-bit key (in hex): a1a2b1b2

After Initial Permutation: 400000e060a020a0

Initial L: 400000e0, R: 60a020a0

Round 1 -> L: 60a020a0, R: 400000e0

Round 2 -> L: 400000e0, R: 60a020a0

Round 3 -> L: 60a020a0, R: 400000e0

Round 4 -> L: 400000e0, R: 60a020a0

Round 5 -> L: 60a020a0, R: 400000e0

Round 6 -> L: 400000e0, R: 60a020a0

Round 7 -> L: 60a020a0, R: 400000e0

Round 8 -> L: 400000e0, R: 60a020a0

Round 9 -> L: 60a020a0, R: 400000e0

Round 10 -> L: 400000e0, R: 60a020a0

Round 11 -> L: 60a020a0, R: 400000e0

Round 12 -> L: 400000e0, R: 60a020a0

Round 13 -> L: 60a020a0, R: 400000e0

Round 14 -> L: 400000e0, R: 60a020a0

Round 15 -> L: 60a020a0, R: 400000e0

Round 16 -> L: 400000e0, R: 60a020a0

Ciphertext: 57c213

Date: 10/ 12 / 2024

Decryption:

Enter 64-bit plaintext (in hex): 57c213

Enter 64-bit key (in hex): a1a2b1b2

After Initial Permutation: 60a020a0400000e0

Initial L: 60a020a0, R: 400000e0

Round 1 -> L: 400000e0, R: 60a020a0

Round 2 -> L: 60a020a0, R: 400000e0

Round 3 -> L: 400000e0, R: 60a020a0

Round 4 -> L: 60a020a0, R: 400000e0

Round 5 -> L: 400000e0, R: 60a020a0

Round 6 -> L: 60a020a0, R: 400000e0

Round 7 -> L: 400000e0, R: 60a020a0

Round 8 -> L: 60a020a0, R: 400000e0

Round 9 -> L: 400000e0, R: 60a020a0

Round 10 -> L: 60a020a0, R: 400000e0

Round 11 -> L: 400000e0, R: 60a020a0

Round 12 -> L: 60a020a0, R: 400000e0

Round 13 -> L: 400000e0, R: 60a020a0

Round 14 -> L: 60a020a0, R: 400000e0

Round 15 -> L: 400000e0, R: 60a020a0

Round 16 -> L: 60a020a0, R: 400000e0

Ciphertext: abc123

Date: 10/ 12 / 2024

AES encryption and decryption:

```
#include <stdio.h>  // for printf
#include <stdlib.h> // for malloc, free
enum errorCode
{
    SUCCESS = 0,
    ERROR_AES_UNKNOWN_KEYSIZE,
    ERROR_MEMORY_ALLOCATION_FAILED,
};
 unsigned char sbox[256] = {
    // 0    1   2    3    4   5    6    7    8  9    A    B  C    D    E    F
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,  // 0
    //remain all …
}; // F

 unsigned char rsbox[256] =
    {0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb, 0x7c, 2b,
0x04, 0x7e, 0xba, 0x77
//remain all….
};
 unsigned char getSBoxValue(unsigned char num);
 unsigned char getSBoxInvert(unsigned char num);
 void rotate(unsigned char *word);
```

Date: 10/ 12 / 2024

```
// Implementation: Rcon
unsigned char Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
    //remain all….
};
unsigned char getRconValue(unsigned char num);
void core(unsigned char *word, int iteration);
enum keySize
{
    SIZE_16 = 16,
    SIZE_24 = 24,
    SIZE_32 = 32
};
void expandKey(unsigned char *expandedKey, unsigned char *key, enum keySize, size_t
expandedKeySize);

void subBytes(unsigned char *state);
void shiftRows(unsigned char *state);
void shiftRow(unsigned char *state, unsigned char nbr);
void addRoundKey(unsigned char *state, unsigned char *roundKey);
unsigned char galois_multiplication(unsigned char a, unsigned char b);
void mixColumns(unsigned char *state);
void mixColumn(unsigned char *column);
void aes_round(unsigned char *state, unsigned char *roundKey);
void createRoundKey(unsigned char *expandedKey, unsigned char *roundKey);
void aes_main(unsigned char *state, unsigned char *expandedKey, int nbrRounds);
```

Date: 10/ 12 / 2024

```
char aes_encrypt(unsigned char *input, unsigned char *output, unsigned char *key, enum keySize size);

void invSubBytes(unsigned char *state);

 void invShiftRows(unsigned char *state);

 void invShiftRow(unsigned char *state, unsigned char nbr);

 void invMixColumns(unsigned char *state);

 void invMixColumn(unsigned char *column);

 void aes_invRound(unsigned char *state, unsigned char *roundKey);

 void aes_invMain(unsigned char *state, unsigned char *expandedKey, int nbrRounds);

 char aes_decrypt(unsigned char *input, unsigned char *output, unsigned char *key, enum keySize size);

 int main(int argc, char *argv[])

{

  int expandedKeySize = 176;

unsigned char expandedKey[expandedKeySize];

    unsigned char key[16] = {'k', 'k', 'k', 'k', 'e', 'e', 'e', 'e', 'y', 'y', 'y', 'y', '.', '.', '.', '.'};

    enum keySize size = SIZE_16;

    unsigned char plaintext[16] = {'a', 'b', 'c', 'd', 'e', 'f', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0'};

    unsigned char ciphertext[16];

    unsigned char decryptedtext[16];

    int i;

    printf("*************************************************\n");

    printf("*   Basic implementation of AES algorithm in C   *\n");

    printf("*************************************************\n");

    printf("\nCipher Key (HEX format):\n");

    for (i = 0; i < 16; i++)

    {

       printf("%2.2x%c", key[i], ((i + 1) % 16) ? ' ' : '\n');
```

Date: 10/ 12 / 2024

```
}
expandKey(expandedKey, key, size, expandedKeySize);
  printf("\nExpanded Key (HEX format):\n");
  for (i = 0; i < expandedKeySize; i++) {  printf("%2.2x%c", expandedKey[i], ((i + 1) % 16) ? ' ' : '\n');    }
  printf("\nPlaintext (HEX format):\n");
  for (i = 0; i < 16; i++)
  {       printf("%2.2x%c", plaintext[i], ((i + 1) % 16) ? ' ' : '\n'); }
   aes_encrypt(plaintext, ciphertext, key, SIZE_16);
   printf("\nCiphertext (HEX format):\n");
   for (i = 0; i < 16; i++)
   {
      printf("%2.2x%c", ciphertext[i], ((i + 1) % 16) ? ' ' : '\n');
   }
   aes_decrypt(ciphertext, decryptedtext, key, SIZE_16);
   printf("\nDecrypted text (HEX format):\n");
  for (i = 0; i < 16; i++)
   {
      printf("%2.2x%c", decryptedtext[i], ((i + 1) % 16) ? ' ' : '\n');
   }
  return 0;
}
unsigned char getSBoxValue(unsigned char num)
{
   return sbox[num];
}
```

Date: 10/ 12 / 2024

```
unsigned char getSBoxInvert(unsigned char num)
{
   return rsbox[num];
}

void rotate(unsigned char *word)
{
   unsigned char c;
   int i;
   c = word[0];
   for (i = 0; i < 3; i++)
      word[i] = word[i + 1];
   word[3] = c;
}
unsigned char getRconValue(unsigned char num)
{
   return Rcon[num];
}
void core(unsigned char *word, int iteration)
{
   int i;
   rotate(word);
   for (i = 0; i < 4; ++i)
   {
      word[i] = getSBoxValue(word[i]);
   }
```

Date: 10/ 12 / 2024

```
    word[0] = word[0] ^ getRconValue(iteration);
}
void expandKey(unsigned char *expandedKey,
               unsigned char *key,
               enum keySize size,
               size_t expandedKeySize)
{
  int currentSize = 0;
    int rconIteration = 1;
    int i;
    unsigned char t[4] = {0}; // temporary 4-byte variable
  for (i = 0; i < size; i++)
       expandedKey[i] = key[i];
    currentSize += size;
     while (currentSize < expandedKeySize)
    {
      for (i = 0; i < 4; i++)
       {  t[i] = expandedKey[(currentSize - 4) + i];       }
       if (currentSize % size == 0)
       {  core(t, rconIteration++)}
     if (size == SIZE_32 && ((currentSize % size) == 16))
       {
          for (i = 0; i < 4; i++)
              t[i] = getSBoxValue(t[i]);
       }
```

Date: 10/ 12 / 2024

```
 for (i = 0; i < 4; i++)
      {    expandedKey[currentSize] = expandedKey[currentSize - size] ^ t[i];
         currentSize++;
      } }
}
void subBytes(unsigned char *state)
{    int i;
   for (i = 0; i < 16; i++)
      state[i] = getSBoxValue(state[i]);
}
 void shiftRows(unsigned char *state)
{    int i;
 for (i = 0; i < 4; i++)
      shiftRow(state + i * 4, i);
}
void shiftRow(unsigned char *state, unsigned char nbr)
{  int i, j;
   unsigned char tmp;
 for (i = 0; i < nbr; i++)
   {
      tmp = state[0];
      for (j = 0; j < 3; j++)
         state[j] = state[j + 1];
      state[3] = tmp;
   }
```

Date: 10/ 12 / 2024

```
}
void addRoundKey(unsigned char *state, unsigned char *roundKey)
{
   int i;
   for (i = 0; i < 16; i++)
      state[i] = state[i] ^ roundKey[i];
}
unsigned char galois_multiplication(unsigned char a, unsigned char b)
{
   unsigned char p = 0;
   unsigned char counter;
   unsigned char hi_bit_set;
   for (counter = 0; counter < 8; counter++)
   {
      if ((b & 1) == 1)
         p ^= a;
      hi_bit_set = (a & 0x80);
      a <<= 1;
      if (hi_bit_set == 0x80)
         a ^= 0x1b;
      b >>= 1;
   }
   return p;
}
void mixColumns(unsigned char *state)
{
```

Date: 10/ 12 / 2024

```
int i, j;
   unsigned char column[4];
   for (i = 0; i < 4; i++)
   {
      for (j = 0; j < 4; j++)
      {
         column[j] = state[(j * 4) + i];
      }
      mixColumn(column);
      for (j = 0; j < 4; j++)
      {  state[(j * 4) + i] = column[j];  }
   } }
void mixColumn(unsigned char *column)
{
   unsigned char cpy[4];
   int i;
   for (i = 0; i < 4; i++)
   {
      cpy[i] = column[i];
   }
   column[0] = galois_multiplication(cpy[0], 2) ^
            galois_multiplication(cpy[3], 1) ^
            galois_multiplication(cpy[2], 1) ^
            galois_multiplication(cpy[1], 3);
   column[1] = galois_multiplication(cpy[1], 2) ^
```

Date: 10/ 12 / 2024

```
        galois_multiplication(cpy[0], 1) ^
            galois_multiplication(cpy[3], 1) ^
            galois_multiplication(cpy[2], 3);
    column[2] = galois_multiplication(cpy[2], 2) ^
            galois_multiplication(cpy[1], 1) ^
            galois_multiplication(cpy[0], 1) ^
            galois_multiplication(cpy[3], 3);
    column[3] = galois_multiplication(cpy[3], 2) ^
            galois_multiplication(cpy[2], 1) ^
            galois_multiplication(cpy[1], 1) ^
            galois_multiplication(cpy[0], 3);
}
void aes_round(unsigned char *state, unsigned char *roundKey)
{
    subBytes(state);
    shiftRows(state);
    mixColumns(state);
    addRoundKey(state, roundKey);
}
void createRoundKey(unsigned char *expandedKey, unsigned char *roundKey)
{
    int i, j;
  for (i = 0; i < 4; i++)
    {
      for (j = 0; j < 4; j++)
```

Date: 10/ 12 / 2024

```
roundKey[(i + (j * 4))] = expandedKey[(i * 4) + j];
  } }
void aes_main(unsigned char *state, unsigned char *expandedKey, int nbrRounds)
{
   int i = 0;
unsigned char roundKey[16];
  createRoundKey(expandedKey, roundKey);
   addRoundKey(state, roundKey);
  for (i = 1; i < nbrRounds; i++)
   {
      createRoundKey(expandedKey + 16 * i, roundKey);
      aes_round(state, roundKey);
   }
  createRoundKey(expandedKey + 16 * nbrRounds, roundKey);
   subBytes(state);
   shiftRows(state);
   addRoundKey(state, roundKey);
}
char aes_encrypt(unsigned char *input,
              unsigned char *output,
              unsigned char *key,
              enum keySize size)
{
 int expandedKeySize;
 int nbrRounds;
  unsigned char *expandedKey;
```

Date: 10/ 12 / 2024

```
unsigned char block[16];
   int i, j;
switch (size)
   {
   case SIZE_16:
      nbrRounds = 10;
      break;
   case SIZE_24:
      nbrRounds = 12;
      break;
   case SIZE_32:
      nbrRounds = 14;
      break;
   default:
      return ERROR_AES_UNKNOWN_KEYSIZE;
      break;
   }
   expandedKeySize = (16 * (nbrRounds + 1));
   expandedKey = (unsigned char *)malloc(expandedKeySize * sizeof(unsigned char));
   if (expandedKey == NULL)
   { return ERROR_MEMORY_ALLOCATION_FAILED; }
   else
   {for (i = 0; i < 4; i++)  {
       for (j = 0; j < 4; j++)
            block[(i + (j * 4))] = input[(i * 4) + j];
     }
```

```
expandKey(expandedKey, key, size, expandedKeySize);

     aes_main(block, expandedKey, nbrRounds);

     for (i = 0; i < 4; i++)

      {

         for (j = 0; j < 4; j++)

             output[(i * 4) + j] = block[(i + (j * 4))];

      }

free(expandedKey);

       expandedKey = NULL;

   }

   return SUCCESS;

}

void invSubBytes(unsigned char *state)

{    int i;

   for (i = 0; i < 16; i++)

       state[i] = getSBoxInvert(state[i]);

}

void invShiftRows(unsigned char *state)

{

   int i;

 for (i = 0; i < 4; i++)

       invShiftRow(state + i * 4, i);

}

void invShiftRow(unsigned char *state, unsigned char nbr)

{

   int i, j;
```

Date: 10/ 12 / 2024

```
unsigned char tmp;
    for (i = 0; i < nbr; i++)
    {
       tmp = state[3];
       for (j = 3; j > 0; j--)
          state[j] = state[j - 1];
       state[0] = tmp;
    }}
 void invMixColumns(unsigned char *state)
{
    int i, j;
    unsigned char column[4];
for (i = 0; i < 4; i++)
    {
  for (j = 0; j < 4; j++)
       {     column[j] = state[(j * 4) + i]; }
      invMixColumn(column);
      for (j = 0; j < 4; j++)
       {state[(j * 4) + i] = column[j];    }
}
 void invMixColumn(unsigned char *column)
{
    unsigned char cpy[4];
    int i;
    for (i = 0; i < 4; i++)
```

Date: 10/ 12 / 2024

```
{   cpy[i] = column[i];  }
   column[0] = galois_multiplication(cpy[0], 14) ^
               galois_multiplication(cpy[3], 9) ^
               galois_multiplication(cpy[2], 13) ^
               galois_multiplication(cpy[1], 11);
   column[1] = galois_multiplication(cpy[1], 14) ^
               galois_multiplication(cpy[0], 9) ^
               galois_multiplication(cpy[3], 13) ^
               galois_multiplication(cpy[2], 11);
   column[2] = galois_multiplication(cpy[2], 14) ^
               galois_multiplication(cpy[1], 9) ^
               galois_multiplication(cpy[0], 13) ^
               galois_multiplication(cpy[3], 11);
   column[3] = galois_multiplication(cpy[3], 14) ^
               galois_multiplication(cpy[2], 9) ^
               galois_multiplication(cpy[1], 13) ^
               galois_multiplication(cpy[0], 11);
}
void aes_invRound(unsigned char *state, unsigned char *roundKey)
{
  invShiftRows(state);
   invSubBytes(state);
   addRoundKey(state, roundKey);
   invMixColumns(state);
}
void aes_invMain(unsigned char *state, unsigned char *expandedKey, int nbrRounds)
```

Date: 10/ 12 / 2024

```
{
    int i = 0;
    unsigned char roundKey[16];
    createRoundKey(expandedKey + 16 * nbrRounds, roundKey);
    addRoundKey(state, roundKey);
    for (i = nbrRounds - 1; i > 0; i--)
    {
        createRoundKey(expandedKey + 16 * i, roundKey);
        aes_invRound(state, roundKey);
    }
    createRoundKey(expandedKey, roundKey);
    invShiftRows(state);
    invSubBytes(state);
    addRoundKey(state, roundKey);
}
 char aes_decrypt(unsigned char *input,
                  unsigned char *output,
                  unsigned char *key,
                  enum keySize size)
{
  int expandedKeySize;
  int nbrRounds;
    unsigned char *expandedKey;
block[16];
    int i, j;
```

Date: 10/ 12 / 2024

```
switch (size)
   {
   case SIZE_16:
      nbrRounds = 10;
      break;
   case SIZE_24:
      nbrRounds = 12;
      break;
   case SIZE_32:
      nbrRounds = 14;
      break;
   default:
      return ERROR_AES_UNKNOWN_KEYSIZE;
      break;
   }
   expandedKeySize = (16 * (nbrRounds + 1));
   expandedKey = (unsigned char *)malloc(expandedKeySize * sizeof(unsigned char));
   if (expandedKey == NULL)
   {  return ERROR_MEMORY_ALLOCATION_FAILED; }
   else
   { for (i = 0; i < 4; i++)
     {
   for (j = 0; j < 4; j++)
            block[(i + (j * 4))] = input[(i * 4) + j];
      }
expandKey(expandedKey, key, size, expandedKeySize);
```

Date: 10/ 12 / 2024

```
 aes_invMain(block, expandedKey, nbrRounds);

for (i = 0; i < 4; i++)

      {

 for (j = 0; j < 4; j++)

            output[(i * 4) + j] = block[(i + (j * 4))];

      }

 free(expandedKey);

      expandedKey = NULL;

   }

   return SUCCESS;

}
```

Date: 10/ 12 / 2024

Output:

```
Cipher Key (HEX format):
6b 6b 6b 6b 65 65 65 65 79 79 79 79 2e 2e 2e 2e

Expanded Key (HEX format):
6b 6b 6b 6b 65 65 65 65 79 79 79 79 2e 2e 2e 2e
5b 5a 5a 5a 3e 3f 3f 3f 47 46 46 46 69 68 68 68
1c 1f 1f a3 22 20 20 9c 65 66 66 da 0c 0e 0e b2
b3 b4 28 5d 91 94 08 c1 f4 f2 6e 1b f8 fc 60 a9
0b 64 fb 1c 9a f0 f3 dd 6e 02 9d c6 96 fe fd 6f
a0 30 53 8c 3a c0 a0 51 54 c2 3d 97 c2 3c c0 f8
6b 8a 12 a9 51 4a b2 f8 05 88 8f 6f c7 b4 4f 97
a6 0e 9a 6f f7 44 28 97 f2 cc a7 f8 35 78 e8 6f
9a 95 32 f9 6d d1 1a 6e 9f 1d bd 96 aa 65 55 f9
cc 69 ab 55 a1 b8 b1 3b 3e a5 0c ad 94 c0 59 54
40 a2 8b 77 e1 1a 3a 4c df bf 36 e1 4b 7f 6f b5

Plaintext (HEX format):
61 62 63 64 65 66 31 32 33 34 35 36 37 38 39 30

Ciphertext (HEX format):
39 62 8b cc c1 cd 48 e4 5f dd b5 e8 9c bf 9d 02

Decrypted text (HEX format):
61 62 63 64 65 66 31 32 33 34 35 36 37 38 39 30
```

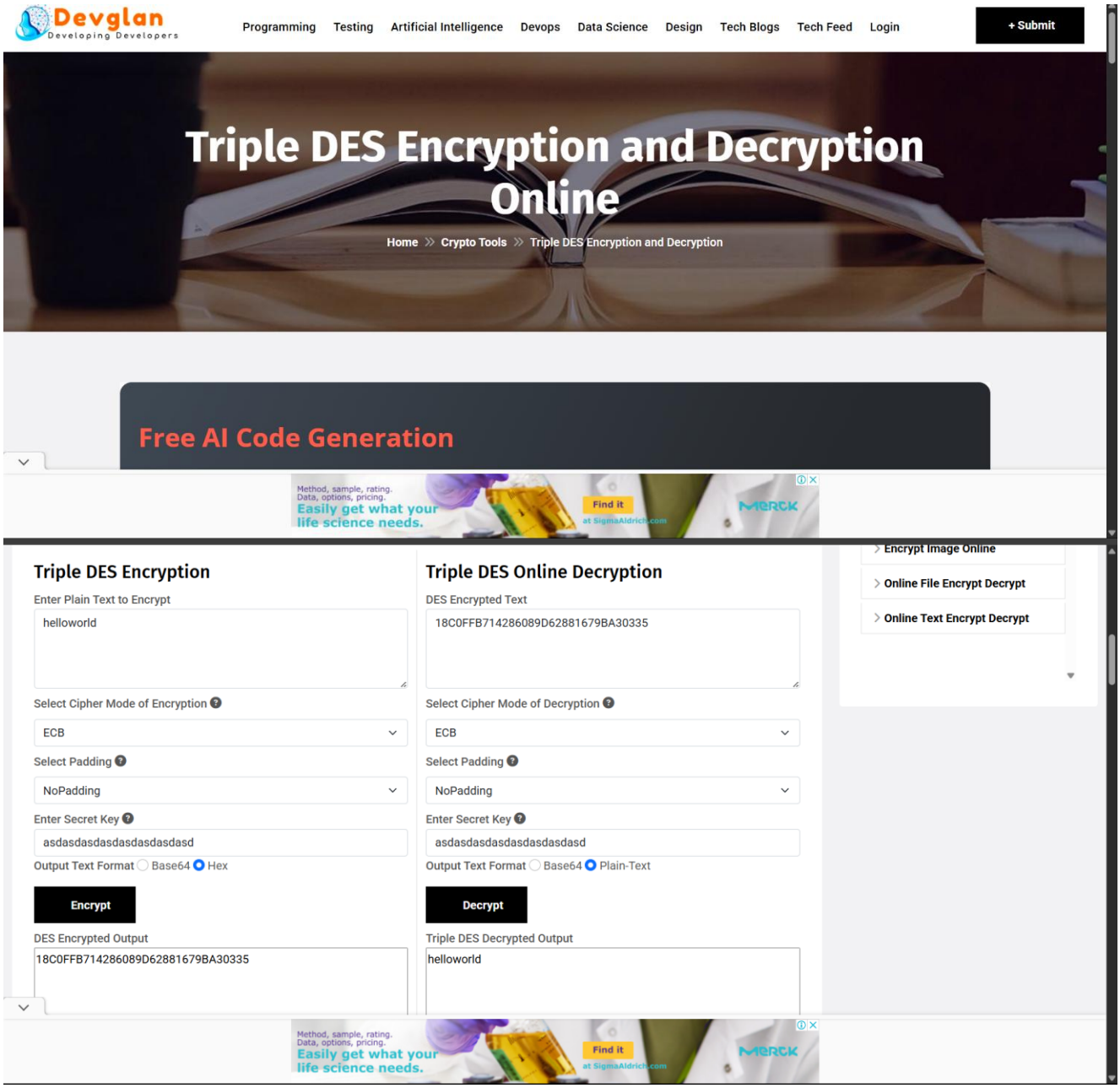
Date: 10/ 12 / 2024

Lab Practical #12:

Simulating the key distribution scenario for Symmetric key Cryptography using the simulator.

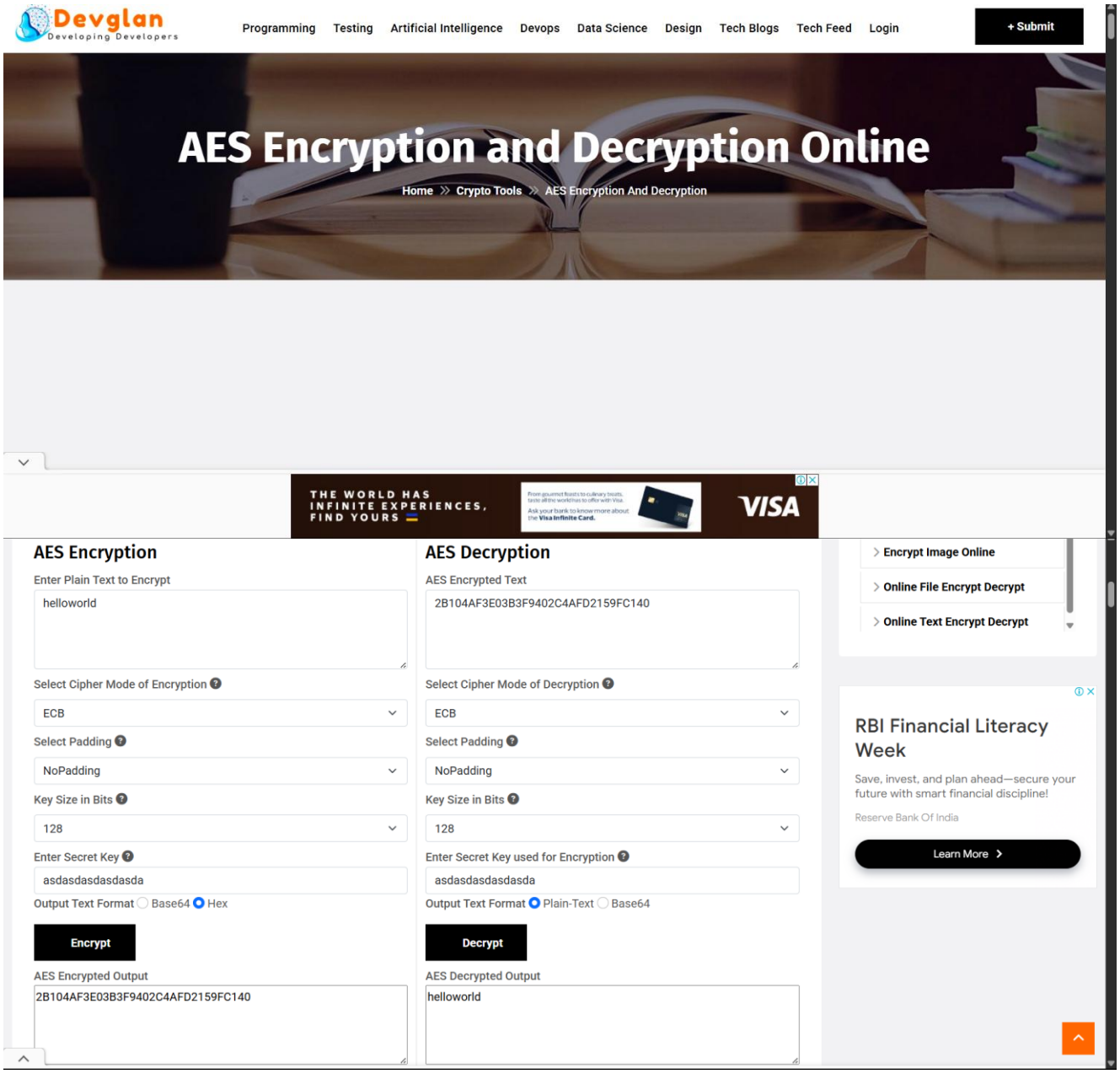
Program:

DES:



Date: 10/ 12 / 2024

AES:



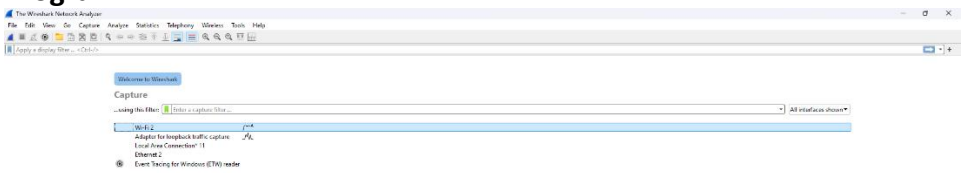
Above tool link: https://www.devglan.com

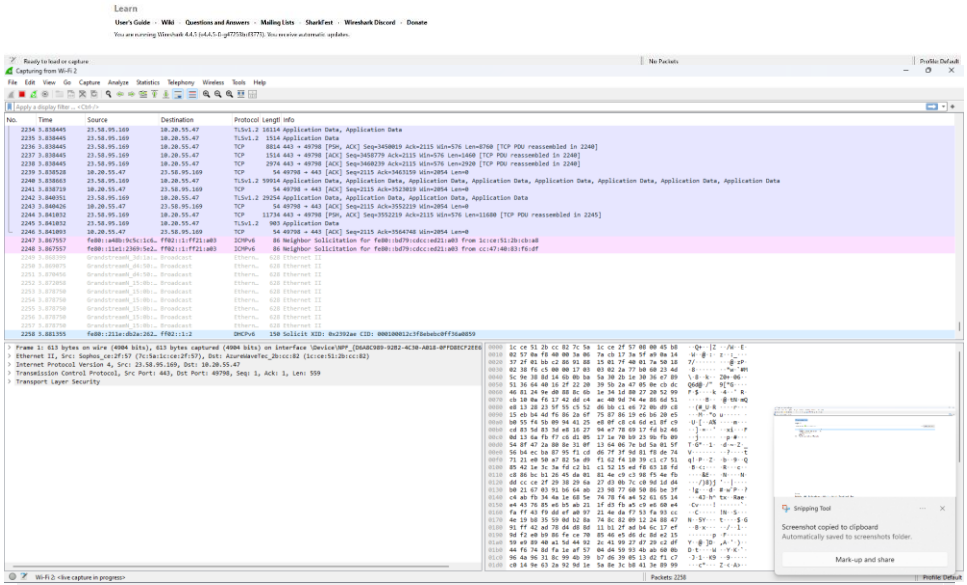
Date: 10/ 12 / 2024

Lab Practical #13:

Use of snort/Wireshark tool for network intrusion detection System to monitor network traffic and analyse attack patterns

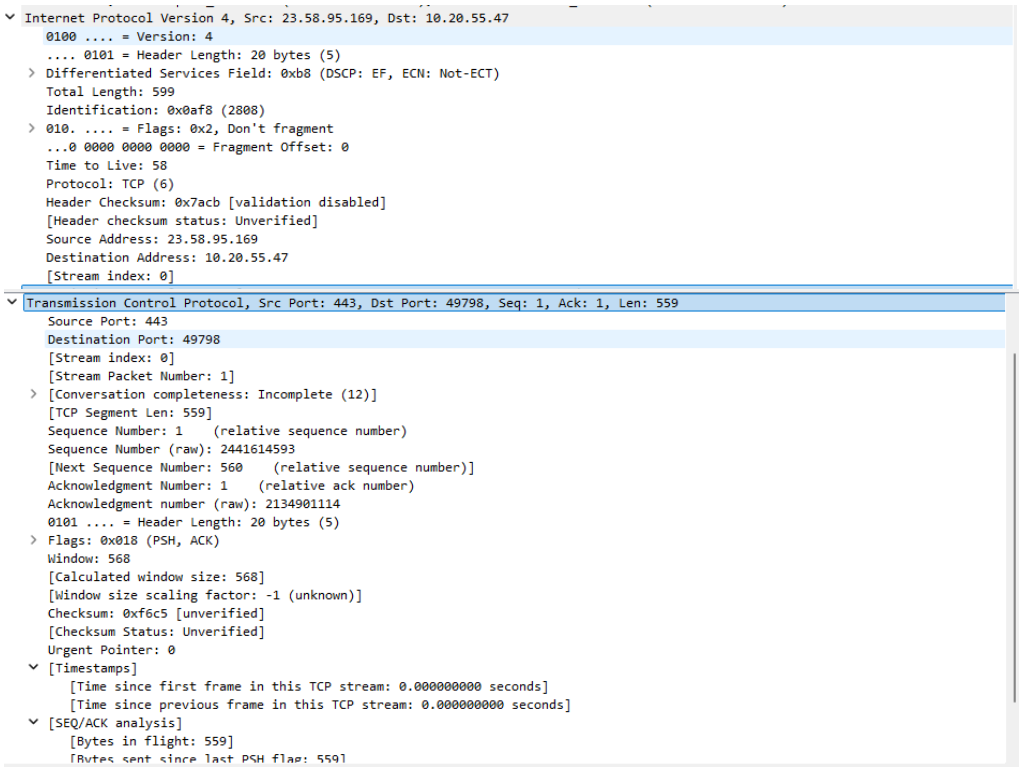
Program:

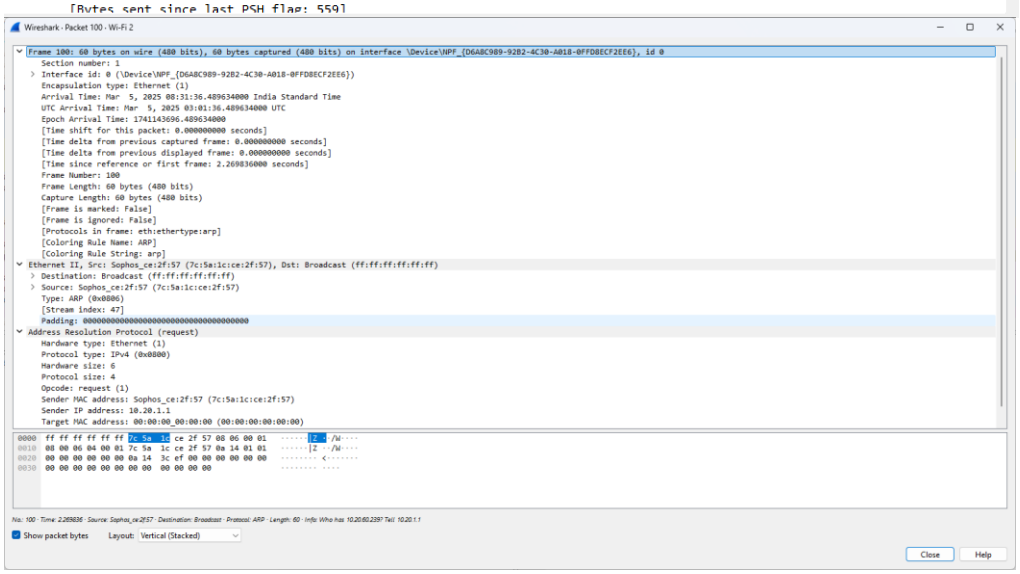




```
▾ Frame 1: 613 bytes on wire (4904 bits), 613 bytes captured (4904 bits) on interface \Device\NPF_{D6A8C989-92B2-4C30-A018-0FFD8ECF2E
    Section number: 1
  › Interface id: 0 (\Device\NPF_{D6A8C989-92B2-4C30-A018-0FFD8ECF2EE6})
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar  5, 2025 08:29:14.072144000 India Standard Time
    UTC Arrival Time: Mar  5, 2025 02:59:14.072144000 UTC
    Epoch Arrival Time: 1741143554.072144000
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 613 bytes (4904 bits)
    Capture Length: 613 bytes (4904 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:tls]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
```

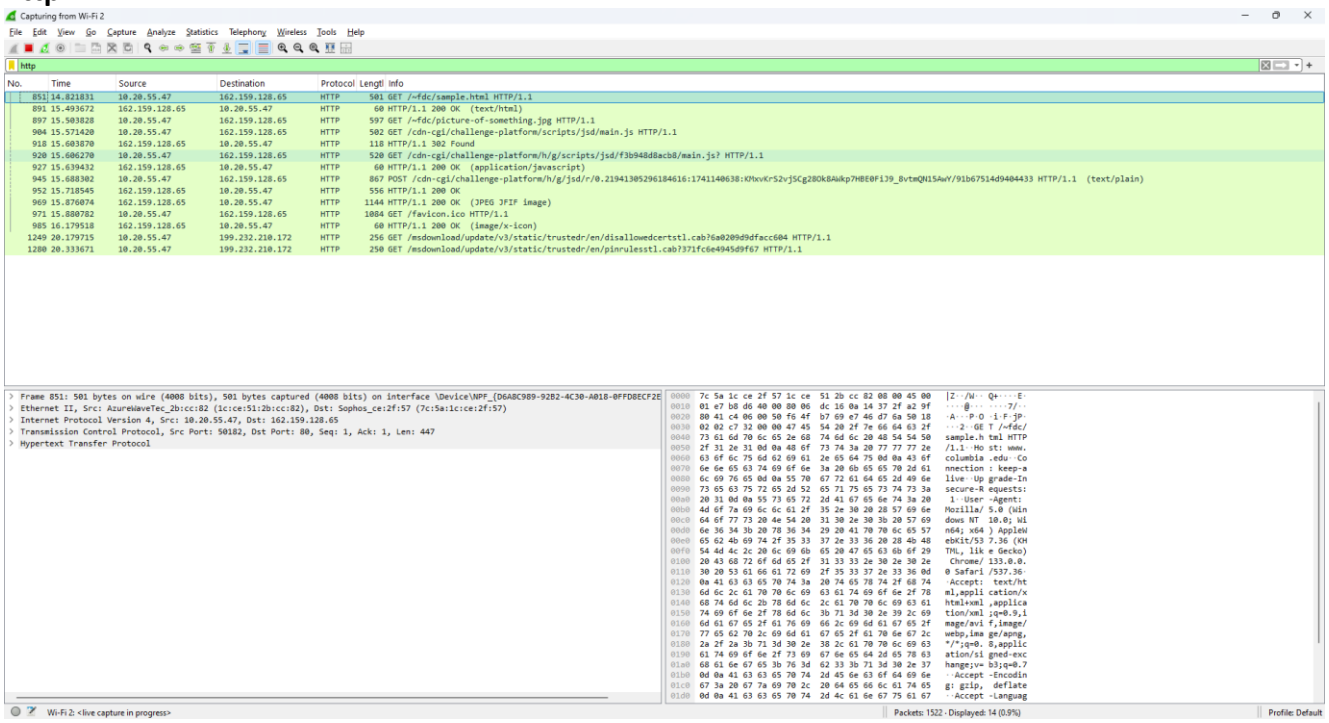
Date: 10/ 12 / 2024





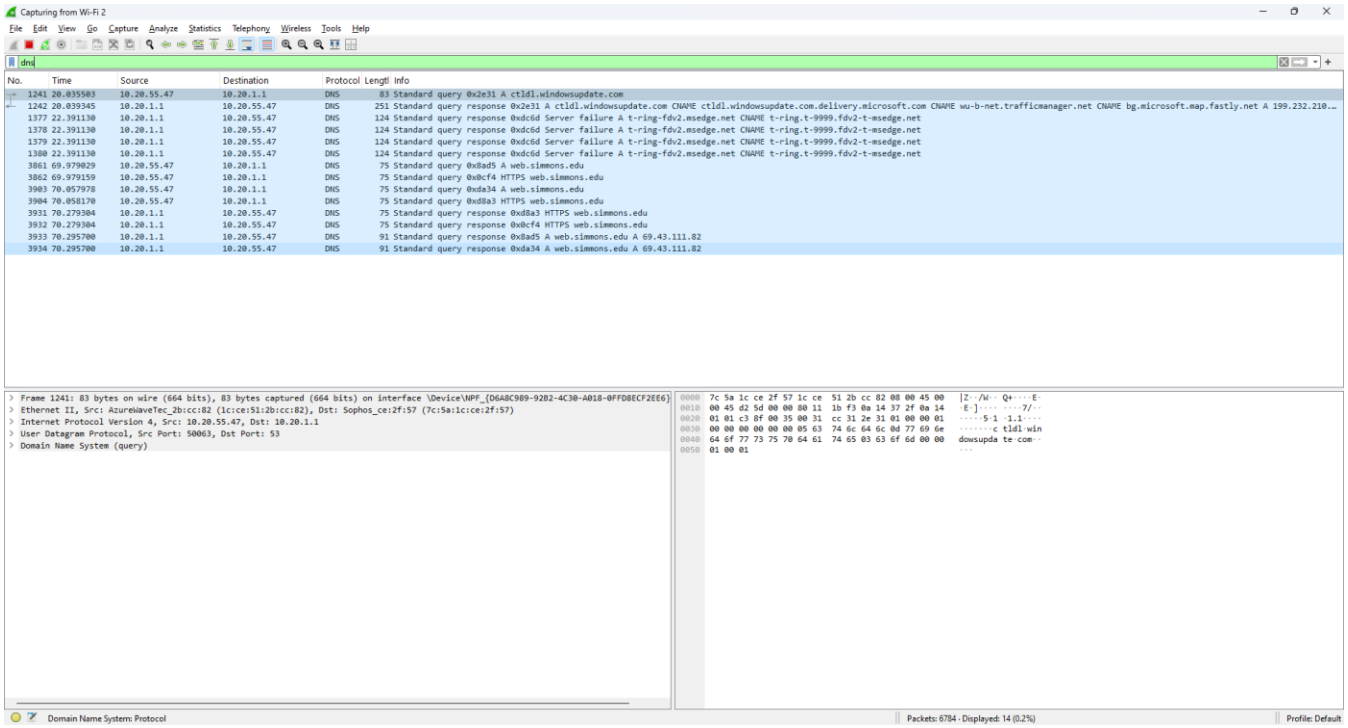
Date: 10/ 12 / 2024

http:



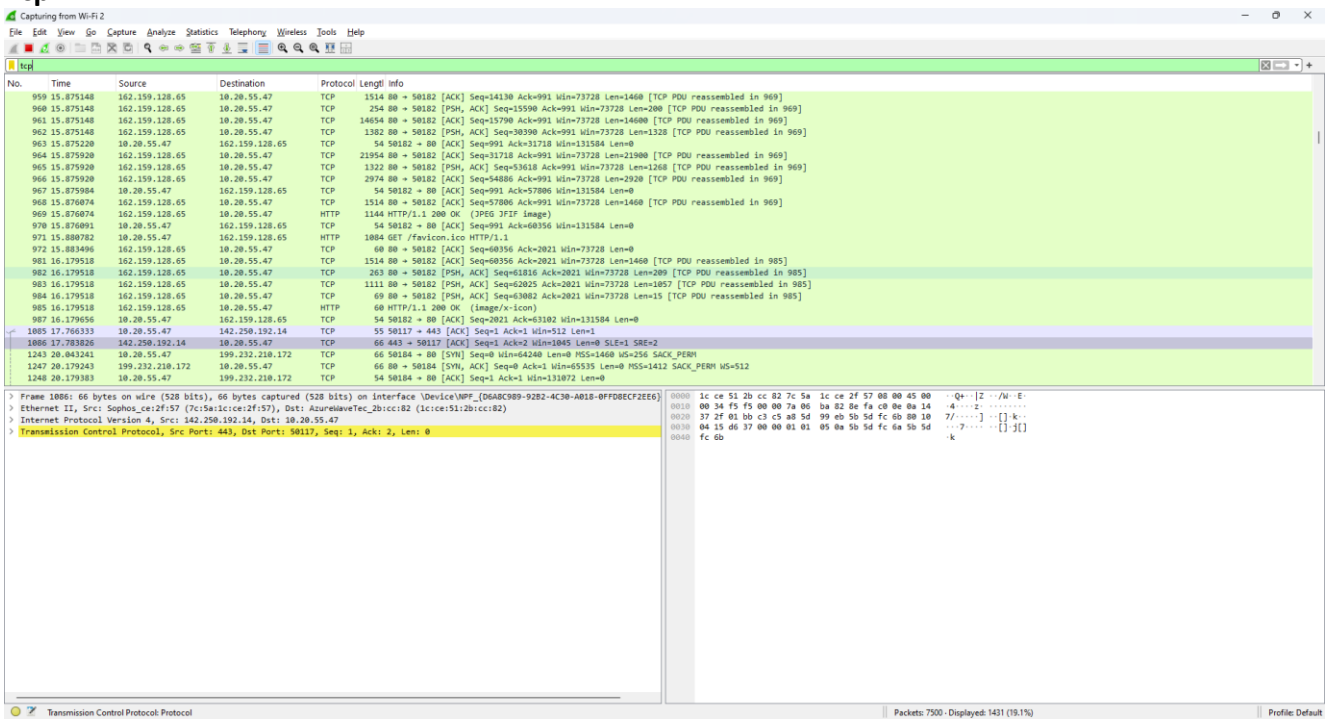
Date: 10/ 12 / 2024

Dns:



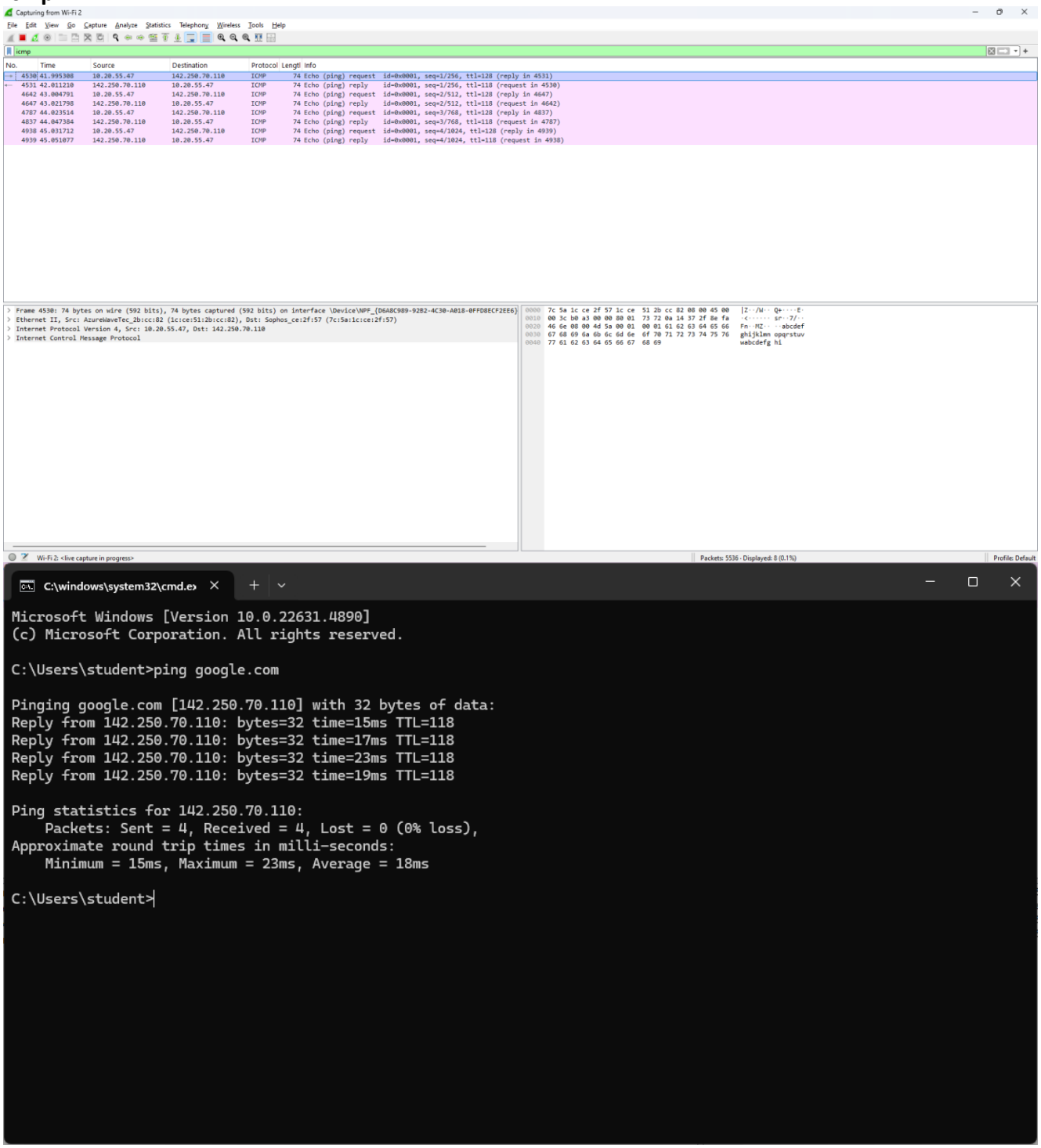
Date: 10/ 12 / 2024

Tcp:



Date: 10/ 12 / 2024

Icmp:



Thank You