LAB 1

Task 1

Title: Caesar Cipher Encryption and Decryption

Algorithm:

Encryption:

1. Accept an integer value K as the shift.
2. Accept the plaintext message.
3. For each character in the message:
   - If the character is an uppercase letter, encrypt using: E(x) = (x + K) % 26, where x is the ASCII value.
   - If the character is a lowercase letter, encrypt using: E(x) = (x + K) % 26, where x is the ASCII value.
   - If the character is not an alphabet letter, keep it unchanged.
4. Print or return the resulting ciphertext.

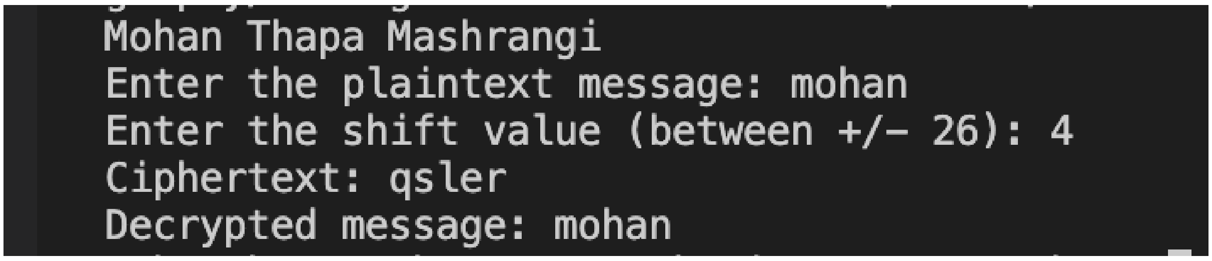
Decryption:

1. Accept an integer value K as the shift.
2. Accept the ciphertext.
3. For each character in the ciphertext:
   - If the character is an uppercase letter, decrypt using: D(x) = (x - K + 26) % 26, where x is the ASCII value.
   - If the character is a lowercase letter, decrypt using: D(x) = (x - K + 26) % 26, where x is the ASCII value.
   - If the character is not an alphabet letter, keep it unchanged.
4. Print or return the resulting plaintext.

Source Code:

```
#include <stdio.h>
#include <ctype.h>
void caesar_cipher(char *message, int shift) {
  for (int i = 0; message[i] != '\0'; i++) {
             if (isalpha(message[i])) {
                     char base = isupper(message[i]) ? 'A' : 'a';
                      message[i] = (message[i] - base + shift + 26) % 26 + base;
          }
  }
}
int main() {
  char message[100];
  int shift;
  printf("Mohan Thapa Mashrangi\n");
  printf("Enter the plaintext message: ");
  fgets(message, 100, stdin);
  printf("Enter the shift value (between +/- 26): ");
  scanf("%d", &shift);
  caesar_cipher(message, shift);
  printf("Ciphertext: %s", message);
  caesar_cipher(message, -shift);
  printf("Decrypted message: %s", message);
  return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Enter the plaintext message: mohan
Enter the shift value (between +/- 26): 4
Ciphertext: qsler
Decrypted message: mohan
```

Conclusion:

Hence, in this way we can implement encryption and decryption using Caesar Cipher in the laboratory.

Task 2

Title: Vigenere Cipher Encryption

Algorithm:

Encryption:

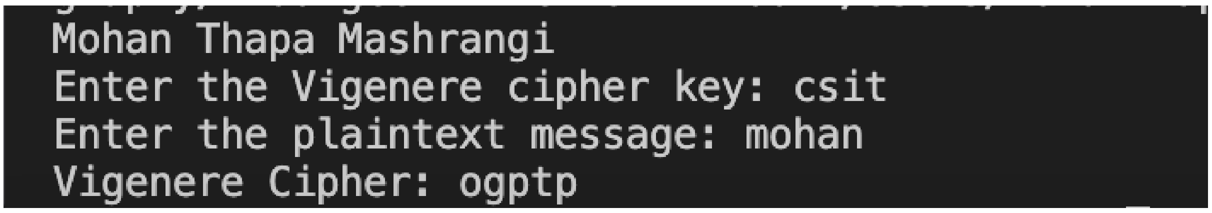
1. Accept a keyword from the user.
2. Accept the plaintext message from the user.
3. Initialise an empty string ciphertext to store the encrypted message.
4. For each character in the plaintext message:
   - If the character is an alphabet letter:
     - Determine the corresponding shift value from the keyword.
     - Encrypt the letter using the Caesar cipher with the calculated shift.
     - Append the encrypted letter to the ciphertext.
   - If the character is not an alphabet letter, keep it unchanged.
5. Print or display the resulting ciphertext.

Source Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void vigenereEncrypt(char *message, char *key) {
   int keyLen = strlen(key);
   int i;
   for (i = 0; *message != '\0'; ++message, ++i) {
      if (isalpha(*message)) {
         char base = isupper(*message) ? 'A' : 'a';
         char keyChar = toupper(key[i % keyLen]);
         *message = ((*message - base + keyChar - 'A') % 26) + base;
      }
   }
}
int main() {
   char key[100];
   char message[100];
   printf("Mohan Thapa Mashrangi\n");
   // Input
```

```
    printf("Enter the Vigenere cipher key: ");
    scanf("%s", key);
    printf("Enter the plaintext message: ");
    scanf(" %[^\n]s", message);
    // Encryption
    vigenereEncrypt(message, key);
    printf("Vigenere Cipher: %s\n", message);
    return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Enter the Vigenere cipher key: csit
Enter the plaintext message: mohan
Vigenere Cipher: ogptp
```

Conclusion:

Hence, in this way we can implement encryption and decryption using Vigenère Cipher in the laboratory.

Task 3

Title: Rail Fence Cipher Encryption

Algorithm:

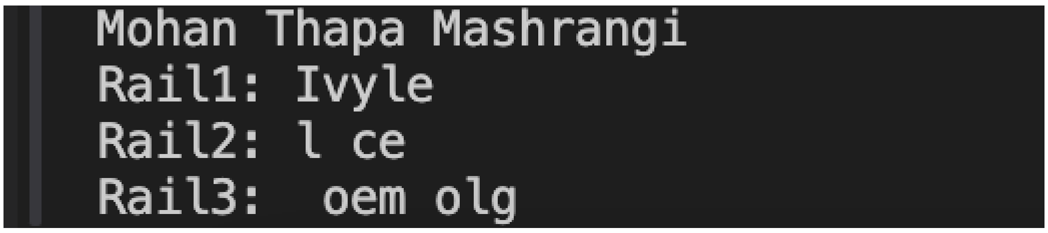
Encryption:

1. Accept a plaintext message from the user.
2. Set the depth of the Rail Fence cipher to 3.
3. Initialize three strings rail1, rail2, and rail3 to represent the three rails.
4. Iterate through each character in the plaintext:
   - Place the character on the appropriate rail based on the current depth.
5. Concatenate the three rail strings to get the ciphertext.
6. Print or display the resulting ciphertext.

Source Code:

```
#include <stdio.h>
#include <string.h>
void railFenceEncrypt(char *message, int depth) {
   int len = strlen(message);
   char rail1[len], rail2[len], rail3[len];
   int rail1Index = 0, rail2Index = 0, rail3Index = 0;
   for (int i = 0; i < len; ++i) {
      if (i % (2 * (depth - 1)) == 0) {
         rail1[rail1Index++] = message[i];
      } else if (i % (depth - 1) == 0) {
         rail2[rail2Index++] = message[i];
      } else {
         rail3[rail3Index++] = message[i];
      }
   }
   rail1[rail1Index] = rail2[rail2Index] = rail3[rail3Index] = '\0';
   printf("Rail1: %s\n", rail1);
   printf("Rail2: %s\n", rail2);
   printf("Rail3: %s\n", rail3);
}
int main() {
   char message[] = "I love my college";
   int depth = 3;
   // Encryption
   railFenceEncrypt(message, depth);
   return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Rail1: Ivyle
Rail2: l ce
Rail3:  oem olg
```

Conclusion:

Hence, in this way we can implement encryption of a given plaintext using Rail Fence Cipher in the laboratory.

Task 4

Title: Initial Permutation in DES Algorithm

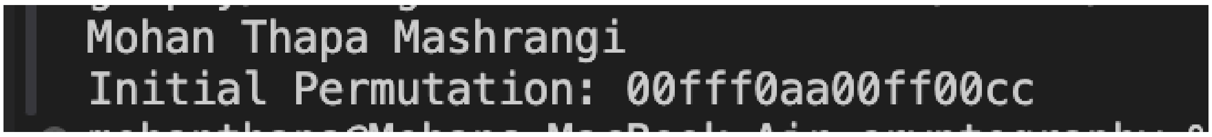
Algorithm:

1. Accept a plaintext message from the user.
2. Define the initial permutation table for DES.
3. Initialise an array initialPermutation[64] to store the permuted bits.
4. Iterate through each bit in the initial permutation table:
    - Calculate the corresponding bit position in the plaintext message.
    - Set the corresponding bit in the initialPermutation array.
5. Print or display the resulting initialPermutation.

Source Code:

```
#include <stdio.h>
#include <stdint.h>
void performInitialPermutation(char *plaintext, int initialPermutationTable[64]) {
   uint64_t initialPermutation = 0;
   for (int i = 0; i < 64; ++i) {
      int bitPosition = initialPermutationTable[i] - 1;
      int byteIndex = bitPosition / 8;
      int bitOffset = bitPosition % 8;
      initialPermutation <<= 1;
      initialPermutation |= (plaintext[byteIndex] >> (7 - bitOffset)) & 1;
   }
   printf("Initial Permutation: %016llx\n", initialPermutation);
}
int main() {
   char plaintext[] = "0123456789ABCDEF";
   int initialPermutationTable[64] = {
      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
   };
   printf("Mohan Thapa Mashrangi\n");
   // Perform Initial Permutation
   performInitialPermutation(plaintext, initialPermutationTable);
   return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Initial Permutation: 00fff0aa00ff00cc
```

Conclusion:

Hence, in this way we can implement the Initial Permutation Step of DES Encryption Algorithm in the laboratory.

LAB 2

Task 1

Title:Write a program to implement the DES key generation process to generate subkeys. Also, show the subkeys generated at each round.

Algorithm:

Subkey Generation:

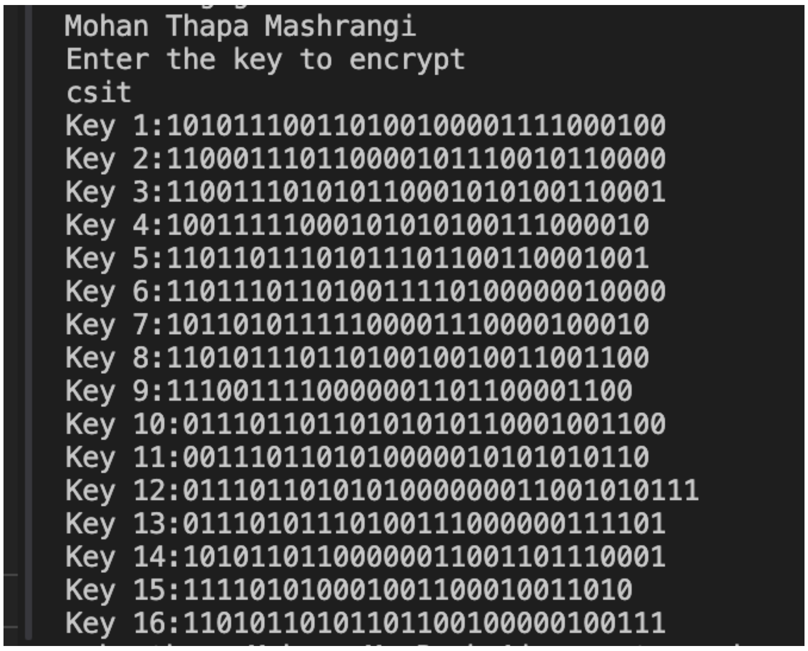
1. Accept a 64-bit key from the user.
2. Perform the PC-1 permutation on the key to obtain a 56-bit value.
3. Split the 56-bit value into left (C) and right (D) halves.
4. For each of the 16 rounds:
    - Determine the shift amount based on the round number.
    - Perform left circular shifts on C and D.
    - Concatenate C and D to form a 56-bit value.
    - Perform the PC-2 permutation on the 56-bit value to obtain a 48-bit subkey.
    - Store and display the subkey for the current round.

Source Code:

```
#include<iostream>
#include<string>
#include<bitset>
using namespace std;

string round_keys[16];
// circular left shift by one
string C_L_Shift_Once(string key_chunk)
{
   string shifted = "";
   for (int i = 1; i < 28; i++)
   {
      shifted += key_chunk[i];
   }
   shifted += key_chunk[0];
   return shifted;
}
// circular left shift by two
string C_L_Shift_Twice(string key_chunk)
{
    string shifted = "";
    for (int i = 0; i < 2; i++)
    {
        for (int j = 1; j < 28; j++)
        {
            shifted += key_chunk[j];
        }
        shifted += key_chunk[0];
        key_chunk = shifted;
        shifted = "";
    }
    return key_chunk;
}
void key_generate(string key)
{
    // initial permutation table to convert the key in 56bits
    int ip[56] = {
        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
        };
    // compression permutation table to compress the key in 48bits
    int cp[48] = {
        14,17,11,24,1,5,
        3,28,15,6,21,10,
        23,19,12,4,26,8,
        16,7,27,20,13,2,
        41,52,31,37,47,55,
        30,40,51,45,33,48,
        44,49,39,56,34,53,
        46,42,50,36,29,32
        };
    // compressing the Key to 56 bit using compression permutation table
    string perm_key ="";
    for(int i = 0; i < 56; i++)
    {
        perm_key+= key[ip[i]-1];
    }
    // dividing the the 56 key into two part
    string left = perm_key.substr(0, 28);
    string right = perm_key.substr(28, 56);

    // generating 16 round key
    for (int i = 0; i < 16; i++)
    {
        // one left circular for 1, 2, 9, 16
        if (i == 0 || i == 1|| i == 8 || i == 15)
        {
            left = C_L_Shift_Once(left);
            right = C_L_Shift_Once(right);
        }
        else
        {
            left = C_L_Shift_Twice(left);
            right = C_L_Shift_Twice(right);
        }
        // key chunks are combined
        string combined_key = left + right;
        string round_key = "";
        for (int i = 0; i < 48; i++)
        {
            round_key += combined_key[cp[i]-1];
        }
        round_keys[i] = round_key;
        cout << "Key "<< i+1 << ":" << round_keys[i] << endl;
    }
}
string TextToBinaryString(string words)
{
    string binaryString = "";
    for (char& _char : words) {
        binaryString +=bitset<8>(_char).to_string();
    }
```

```
        return binaryString;
    }
    int main()
    {
        string key, Plain_Text, key_bin;
        cout << "Enter the key to encrypt" << endl;
        cin >> key;
        key_bin = TextToBinaryString(key).substr(0, 64);
        key_generate(key_bin);
    }
}
```

Output:



```
Mohan Thapa Mashrangi
Enter the key to encrypt
csit
Key 1:101011100110100100001111000100
Key 2:110001110110000101110010110000
Key 3:110011101010110001010100110001
Key 4:10011111000101010100111000010
Key 5:11011011101011101100110001001
Key 6:110111011010011110100000010000
Key 7:10110101111100001110000100010
Key 8:11010111011010010010011001100
Key 9:1110011110000001101100001100
Key 10:01110110110101010110001001100
Key 11:0011101101010000010101010110
Key 12:0111011010101000000011001010111
Key 13:01110101110100111000000111101
Key 14:10101101100000011001101110001
Key 15:111101010001001100010011010
Key 16:11010110101101100100000100111
```

Conclusion:

Hence, in this way we can use and implement DES key generation in the laboratory.

Task 2:

Title:Write a program to apply the round function to a given 32-bit data and subkey, and display the intermediate results.

Algorithm:

1. Accept a 32-bit data and a 48-bit subkey for a specific round.
2. Expand the 32-bit data to 48 bits using the E-Box permutation.
3. XOR the expanded data with the subkey.
4. Split the result into 8 groups of 6 bits.
5. Apply the S-Box substitutions to each 6-bit group.
6. Concatenate the results from the S-Boxes.
7. Perform the P-Box permutation on the concatenated result.
8. XOR the permuted result with the original 32-bit data.
9. The final 32-bit result is the output of the round function.

Source Code:

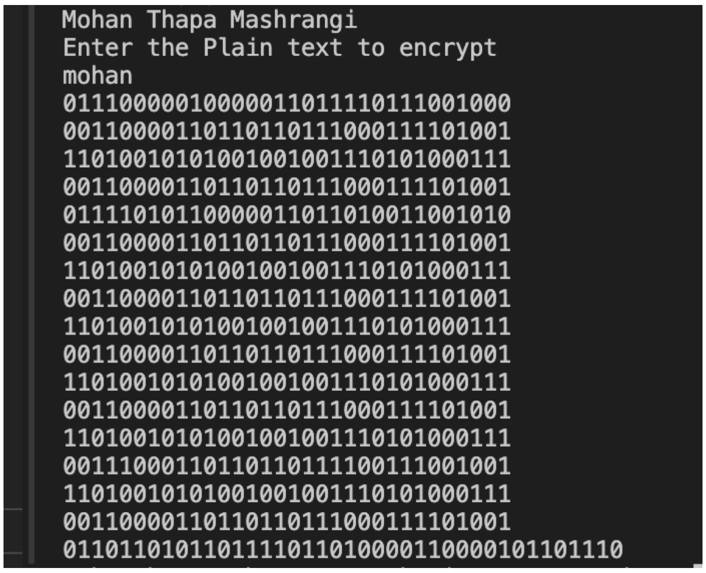
```
#include<iostream>
#include<string>
#include<bitset>
#include<cmath>
using namespace std;
string convertDecimalToBinary(int decimal)
{
   string binary;
   while (decimal != 0)
   {
      if (decimal % 2 == 0) {
         binary = "0" + binary;
      }
      else
      {
         binary = "1" + binary;
      }
      decimal = decimal / 2;
   }
   while(binary.length() < 4){
      binary = "0" + binary;
   }
   return binary;
```

```
}
int convertBinaryToDecimal(string binary)
{
   int decimal = 0;
   int counter = 0;
   int size = binary.length();
   for(int i = size-1; i >= 0; i--)
   {
      if(binary[i] == '1'){
         decimal += pow(2, counter);
      }
   counter++;
   }
   return decimal;
}

string Xor(string a, string b){
   string result = "";
   int size = b.size();
   for(int i = 0; i < size; i++){
      if(a[i] != b[i]){
         result += "1";
      }
      else{
         result += "0";
      }
   }
   return result;
}

void round_function(string Right_Plain_text, const string Round_key) {
   int expansion_table[48] = {
      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
      };
```

```
int substition_boxes[8][4][16]=
{{
   14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
   0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
   4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
   15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13
},
{
   15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
   3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
   0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
   13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9
},
{
   10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
   13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
   13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
   1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12
},
{
   7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
   13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
   10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
   3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14
},
{
   2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
   14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
   4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
   11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3
},
{
   12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
   10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
   9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
   4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13
},
{
        4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
        13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
        1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
        6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12
    },
    {
        13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
        1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
        7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
        2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11
    }};

    // The permutation table
    int permutation_tab[32] = {
    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
    };
    // Apply the expansion permutation.The right half of the plain text is expanded
    string right_expanded="";
    for(int i = 0; i < 48; i++) {
        right_expanded += Right_Plain_text[expansion_table[i]];
    }

    // XOR with the round key.
    string xored = Xor(Round_key, right_expanded);
    string res = "";

    // Apply the S-boxes.
    string S_box_outputs(32, '\0');
    for (int i = 0; i < 8; i++) {
    string row1= xored.substr(i*6,1) + xored.substr(i*6 + 5,1);
            int row = convertBinaryToDecimal(row1);
            string col1 = xored.substr(i*6 + 1,1) + xored.substr(i*6 + 2,1) +
xored.substr(i*6 + 3,1) + xored.substr(i*6 + 4,1);;
            int col = convertBinaryToDecimal(col1);
            int val = substition_boxes[i][row][col];
            res += convertDecimalToBinary(val);
    }

  // Apply the P-box permutation.
    string perm2 ="";
    for(int i = 0; i < 32; i++){
        perm2 += res[permutation_tab[i]-1];
    }
    cout << perm2 <<endl;
}

string TextToBinaryString(string words)
{
    string binaryString = "";
    for (char& _char : words) {
        binaryString +=bitset<8>(_char).to_string();
    }
    return binaryString;
}

int main()
{
    string key, Plain_Text, key_bin, binary_plaintext, ciphertext;
    cout << "Mohan Thapa Mashrangi";
    cout << "Enter the Plain text to encrypt" << endl;
    cin >> Plain_Text;
    binary_plaintext = TextToBinaryString(Plain_Text).substr(0, 64);
    string left_half = binary_plaintext.substr(0, 32);
    string right_half = binary_plaintext.substr(32, 32);
    for (int i = 0; i < 16; i++)
    {
        // Get the round key.
        string round_key = "";
        for (int j = 0; j < 48; j++)
        {
            round_key += binary_plaintext[48 * i + j];
        }
        // Apply the round function.
        round_function(right_half, round_key);
        // Swap the left and right halves.
```

```
            string temp = left_half;
            left_half = right_half;
            right_half = temp;
        }
        ciphertext = left_half + right_half;
        cout << ciphertext << endl;
    }
```

Output:



```
Mohan Thapa Mashrangi
Enter the Plain text to encrypt
mohan
01110000010000011011110111001000
00110000110110110111000111101001
11010010101001001001110101000111
00110000110110110111000111101001
01111010110000011011010011001010
00110000110110110111000111101001
11010010101001001001110101000111
00110000110110110111000111101001
11010010101001001001110101000111
00110000110110110111000111101001
11010010101001001001110101000111
00110000110110110111000111101001
11010010101001001001110101000111
00111000110110110111100111001001
11010010101001001001110101000111
00110000110110110111000111101001
0110110101101111011010000110000101101110
```

Conclusion:

Hence, in this way we can use and implement DES round functions in the laboratory.

Task 3:

Title:Implement the IDEA key scheduling algorithm to generate subkeys from the main encryption key

Algorithm:

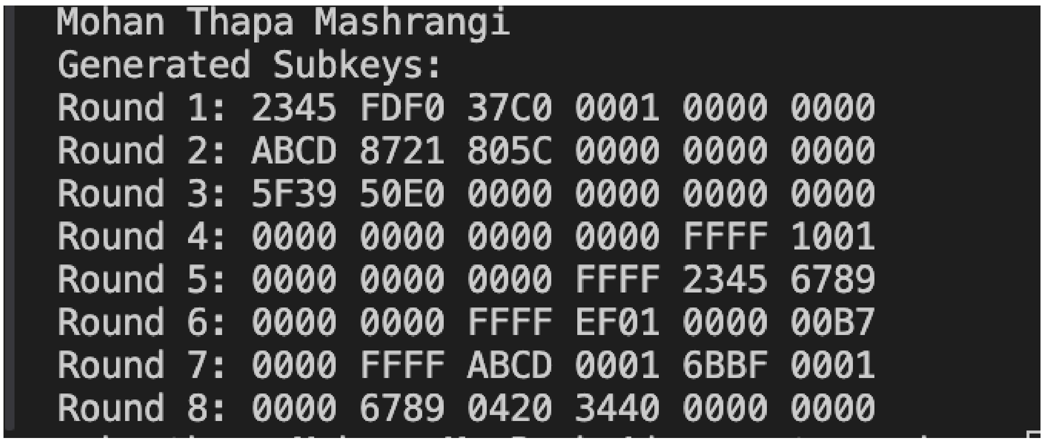
1. Accept a 128-bit main encryption key.
2. Split the 128-bit key into eight 16-bit sub-blocks (K1 to K8).
3. Perform a cyclic left shift on the key, resulting in eight modified sub-blocks (L1 to L8).
4. Concatenate the modified sub-blocks to form the next round key.
5. Repeat steps 3 and 4 for a total of 52 rounds to generate 52 subkeys.

Source Code:

```
#include <stdio.h>
#include <stdint.h>
// Function to perform the key scheduling and generate subkeys
void generateSubkeys(uint16_t* key, uint16_t subkeys[8][6]) {
    int round, subkey;
    uint16_t temp, Z[52];
    // Initialize Z values
    for (round = 0, subkey = 0; round < 8; round++) {
        for (subkey = 0; subkey < 6; subkey++) {
            Z[round * 6 + subkey] = *key;
            key++;
        }
    }
    // Generate subkeys
    for (round = 0; round < 8; round++) {
        for (subkey = 0; subkey < 6; subkey++) {
         subkeys[round][subkey] = Z[(round + subkey) % 8 * 6 + (subkey + 1) % 6];
        }
    }
}
```

```
int main() {
uint16_t mainKey[8] = {0x1001, 0x2345, 0x6789, 0xabcd, 0xef01, 0x2345, 0x6789,
0xabcd};
 uint16_t subkeys[8][6];
 generateSubkeys(mainKey, subkeys);
  printf("Mohan Thapa Mashrangi\n");
    // Display the generated subkeys
    printf("Generated Subkeys:\n");
    for (int round = 0; round < 8; round++) {
        printf("Round %d: ", round + 1);
        for (int subkey = 0; subkey < 6; subkey++) {
            printf("%04X ", subkeys[round][subkey]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Generated Subkeys:
Round 1: 2345 FDF0 37C0 0001 0000 0000
Round 2: ABCD 8721 805C 0000 0000 0000
Round 3: 5F39 50E0 0000 0000 0000 0000
Round 4: 0000 0000 0000 0000 FFFF 1001
Round 5: 0000 0000 0000 FFFF 2345 6789
Round 6: 0000 0000 FFFF EF01 0000 00B7
Round 7: 0000 FFFF ABCD 0001 6BBF 0001
Round 8: 0000 6789 0420 3440 0000 0000
```

Conclusion:

Hence, in this way we can use and implement IDEA key generation in the laboratory.

Task 4:

Title: Write a program to implement the AES SubBytes and ShiftRows operations for encryption. Apply these operations to a given state matrix and show the results.

Theory:

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time. The number of rounds depends on the key length as follows:

- 128 bit key – 10 rounds
- 192 bit key – 12 rounds
- 256 bit key – 14 rounds

SubBytes:

This step implements the substitution. In this step each byte is substituted by another byte. It is performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16-byte (4 x 4) matrix like before.

ShiftRows:

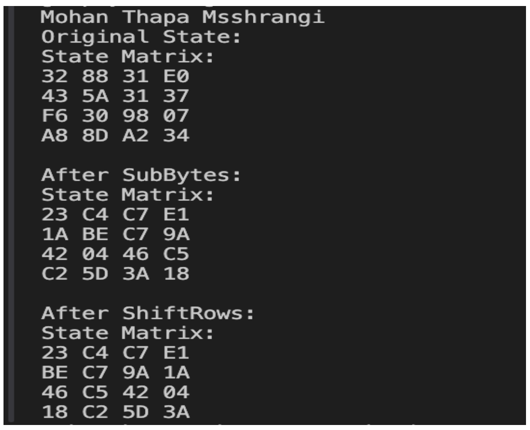
This step is just as it sounds. Each row is shifted a particular number of times.

- The first row is not shifted
- The second row is shifted once to the left.
- The third row is shifted twice to the left.
- The fourth row is shifted thrice to the left.

Source Code:

```
#include <stdio.h>
#include <stdint.h>
// AES S-Box for SubBytes operation
// AES S-Box for SubBytes operation
static const uint8_t sBox[256] = {
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
    0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC,
    0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A,
    0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0,
    0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B,
    0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85,
    0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5,
    0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17,
    0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88,
    0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C,
    0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9,
    0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6,
    0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E,
    0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94,
    0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68,
    0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
};

// AES ShiftRows operation
void shiftRows(uint8_t state[4][4]) {
    uint8_t temp;
    // Shift second row left by 1 byte
    temp = state[1][0];
    state[1][0] = state[1][1];
    state[1][1] = state[1][2];
    state[1][2] = state[1][3];
    state[1][3] = temp;

    // Shift third row left by 2 bytes
    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    // Shift fourth row left by 3 bytes
    temp = state[3][3];
    state[3][3] = state[3][2];
    state[3][2] = state[3][1];
    state[3][1] = state[3][0];
    state[3][0] = temp;
}

// AES SubBytes operation
void subBytes(uint8_t state[4][4]) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = sBox[state[i][j]];
        }
    }
}

// Display the state matrix
void displayState(uint8_t state[4][4]) {
    printf("State Matrix:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02X ", state[i][j]);
        }
        printf("\n");
    }
}

int main() {
    // Example state matrix (4x4)
    uint8_t state[4][4] = {
        {0x32, 0x88, 0x31, 0xe0},
        {0x43, 0x5a, 0x31, 0x37},
        {0xf6, 0x30, 0x98, 0x07},
        {0xa8, 0x8d, 0xa2, 0x34}
    };
  printf("Mohan Thapa Mashrangi\n");
    printf("Original State:\n");
    displayState(state);

    subBytes(state);
    printf("\nAfter SubBytes:\n");
    displayState(state);

    shiftRows(state);
    printf("\nAfter ShiftRows:\n");
    displayState(state);
    return 0;
}
```

Output:



```
Mohan Thapa Msshrangi
Original State:
State Matrix:
32 88 31 E0
43 5A 31 37
F6 30 98 07
A8 8D A2 34

After SubBytes:
State Matrix:
23 C4 C7 E1
1A BE C7 9A
42 04 46 C5
C2 5D 3A 18

After ShiftRows:
State Matrix:
23 C4 C7 E1
BE C7 9A 1A
46 C5 42 04
18 C2 5D 3A
```

Conclusion:

Hence, in this way we can implement AES SubBytes and ShiftRows in the laboratory.

Task 5:

Title: Write a program to implement the AES MixColumns operation for encryption. Apply the operation to a given state matrix and round key, and show the results.

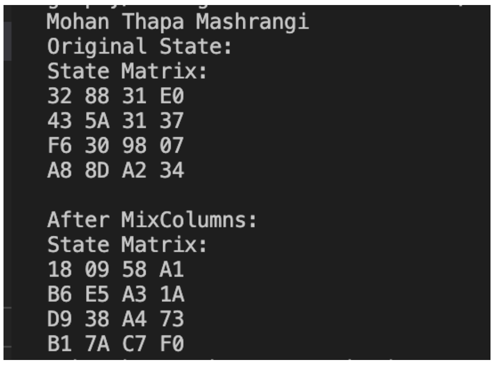
Theory:

AES MixColumns step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

Source Code:

```
#include <stdio.h>
#include <stdint.h>
// AES MixColumns operation
void mixColumns(uint8_t state[4][4]) {
   uint8_t tmp[4];
   for (int c = 0; c < 4; c++) {
      for (int i = 0; i < 4; i++) {
         tmp[i] = state[i][c];
      }
      state[0][c] = (uint8_t)(tmp[0] ^ tmp[1] ^ tmp[2] ^ tmp[3]);
      uint8_t t = tmp[0] ^ tmp[1];
      state[0][c] ^= 0x02 * tmp[0] ^ 0x03 * t;
      state[1][c] = (uint8_t)(t ^ tmp[2] ^ tmp[3]);
      t = tmp[1] ^ tmp[2];
      state[1][c] ^= 0x02 * tmp[1] ^ 0x03 * t;
      state[2][c] = (uint8_t)(t ^ tmp[0] ^ tmp[3]);
      t = tmp[2] ^ tmp[3];
      state[2][c] ^= 0x02 * tmp[2] ^ 0x03 * t;
      state[3][c] = (uint8_t)(t ^ tmp[1] ^ tmp[0]);
      t = tmp[3] ^ tmp[0];
      state[3][c] ^= 0x02 * tmp[3] ^ 0x03 * t;
   }
}
// Display the state matrix
void displayState(uint8_t state[4][4]) {
    printf("State Matrix:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02X ", state[i][j]);
        }
        printf("\n");
    }
}
int main() {
    // Example state matrix (4x4)
    uint8_t state[4][4] = {
        {0x32, 0x88, 0x31, 0xe0},
        {0x43, 0x5a, 0x31, 0x37},
        {0xf6, 0x30, 0x98, 0x07},
        {0xa8, 0x8d, 0xa2, 0x34}
    };

    // Example round key (4x4)
    uint8_t roundKey[4][4] = {
        {0x2b, 0x28, 0xab, 0x09},
        {0x7e, 0xae, 0xf7, 0xcf},
        {0x15, 0xd2, 0x15, 0x4f},
        {0x16, 0xa6, 0x88, 0x3c}
    };
  printf("Mohan Thapa Mashrangi\n");
    printf("Original State:\n");
    displayState(state);

    // Apply MixColumns operation
    mixColumns(state);
    printf("\nAfter MixColumns:\n");
    displayState(state);
    return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Original State:
State Matrix:
32 88 31 E0
43 5A 31 37
F6 30 98 07
A8 8D A2 34

After MixColumns:
State Matrix:
18 09 58 A1
B6 E5 A3 1A
D9 38 A4 73
B1 7A C7 F0
```

Conclusion:

Hence, in this way we can observe and implement AES MixColumns step in the laboratory.

Lab-3

Task 1:

Title: Write a program to implement the Miller-Rabin primality test. Test it with various values of 'n'.

Theory:

The Miller-Rabin primality test or Rabin Miller primality test is a probabilistic primality test. It determines whether a given number is likely to be prime or not. For a number n, the steps for Rabin-Miller Primality test are as follows:

Step 1. Find $n - 1 = 2^k * m$, where m is an odd number

Step 2. Choose a such that $1 < a < n – 1$

Step 3. Compute $b_0 = a^m \bmod n$, …, $b_n = b^2_{n-1} \bmod n$

Step 4. If b_i is +1 the number is composite and id b is -1 the number is probably prime.

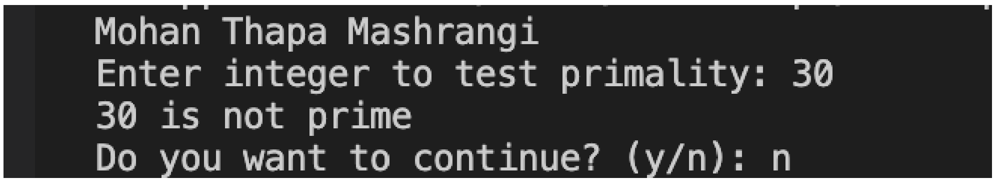
Source Code:

```
#include <iostream>
#include <stdlib.h>
using namespace std;
long long mulmod(long long, long long, long long);
long long modulo(long long, long long, long long);
bool Miller(long long, int);
int main()
{
   do
   {
      int iteration = 10;
      long long num;
      cout << "Mohan Thapa Mashrangi" << endl;
      cout << "Enter integer to test primality: ";
      cin >> num;
      if (Miller(num, iteration))
         cout << num << " is prime" << endl;
```

```
        else
            cout << num << " is not prime" << endl;
        char choice;
        cout << "Do you want to continue? (y/n): ";
        cin >> choice;
        if (choice == 'n' || choice == 'N')
            break;
    } while (true);
    return 0;
}
long long mulmod(long long a, long long b, long long m)
{
    long long x = 0,
              y = a % m;
    while (b > 0)
    {
        if (b % 2 == 1)
        {
            x = (x + y) % m;
        }
        y = (y * 2) % m;
        b /= 2;
    }
    return x % m;
}
long long modulo(long long base, long long e, long long m)
{
    long long x = 1;
    long long y = base;
    while (e > 0)
    {
        if (e % 2 == 1)
            x = (x * y) % m;
        y = (y * y) % m;
        e = e / 2;
    }
    return x % m;
}
bool Miller(long long p, int iteration)
{
    if (p < 2)
    {
        return false;
    }
    if (p != 2 && p % 2 == 0)
    {
        return false;
    }
    long long s = p - 1;
    while (s % 2 == 0)
    {
        s /= 2;
    }
    for (int i = 0; i < iteration; i++)
    {
        long long a = rand() % (p - 1) + 1, temp = s;
        long long mod = modulo(a, temp, p);
        while (temp != p - 1 && mod != 1 && mod != p - 1)
        {
            mod = mulmod(mod, mod, p);
            temp *= 2;
        }
        if (mod != p - 1 && temp % 2 == 0)
        {
            return false;
        }
    }
    return true;
```

```
        }
```

Output:



```
Mohan Thapa Mashrangi
Enter integer to test primality: 30
30 is not prime
Do you want to continue? (y/n): n
```

Conclusion:

Hence, in this way we can use and implement Miller-Rabin Primality test in the laboratory.

Task 2:

Title: Calculate φ(n) (Euler's Totient Function) for a given positive integer 'n.' Verify its correctness for multiple values of 'n.'

Theory:

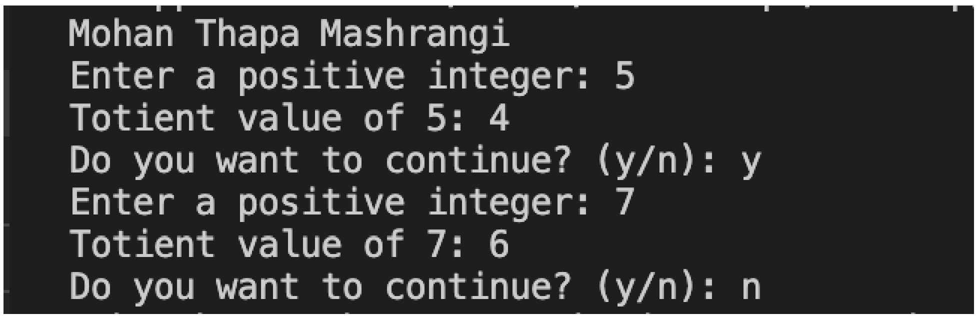
Euler's Totient Function, denoted as φ(n) (phi of n), is a mathematical function that counts the number of positive integers less than or equal to 'n' that are relatively prime to 'n.' In other words, it calculates the count of integers 'k' such that 1 <= k <= n, and gcd(n, k) = 1 (where gcd denotes the greatest common divisor).

Source Code:

```
#include <iostream>
using namespace std;
void computeTotient(int);
int main()
{
   int n;
  cout<<"Mohan Thapa Mashrangi";
   do
   {
      cout << "Enter a positive integer: ";
      cin >> n;
      computeTotient(n);
      cout << "Do you want to continue? (y/n): ";
      char ch;
      cin >> ch;
      if (ch == 'n' || ch == 'N')
         break;

   } while (true);
   return 0;
}

void computeTotient(int n)
```

```
{
    long long phi[n + 1];
    for (int i = 1; i <= n; i++)
        phi[i] = i;
    for (int p = 2; p <= n; p++)
    {
        if (phi[p] == p)
        {
            phi[p] = p - 1;
            for (int i = 2 * p; i <= n; i += p)
            {
                phi[i] = (phi[i] / p) * (p - 1);
            }
        }
    }
    cout << "Totient value of " << n << ": " << phi[n] << endl;
}
```

Output:



```
Mohan Thapa Mashrangi
Enter a positive integer: 5
Totient value of 5: 4
Do you want to continue? (y/n): y
Enter a positive integer: 7
Totient value of 7: 6
Do you want to continue? (y/n): n
```

Conclusion:

Hence, in this way we can calculate Euler Totient Function in the laboratory.

Task 3:

Title: Write a program to apply Fermat's Little Theorem to check if a given number, is a probable prime.

Theory:

Fermat’s little theorem states that if p is a prime number, then for any integer a, the number a^{p-a} is an integer multiple of p.

P is prime and ‘a’ is a positive integer not divisible by P then

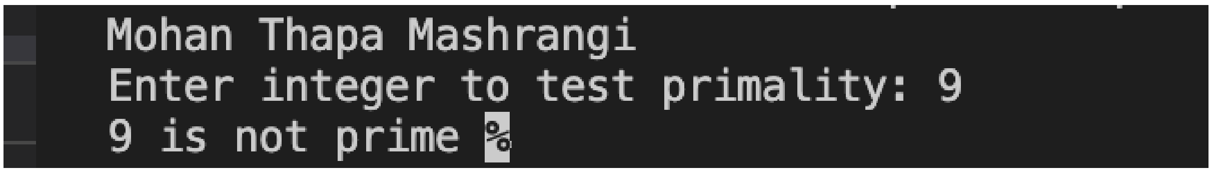
$$a^{p-1} = 1 \bmod p$$

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#define ll long long
ll modulo(ll base, ll exponent, ll mod) {
   ll x = 1;
   ll y = base;
   while (exponent > 0) {
      if (exponent % 2 == 1)
         x = (x * y) % mod;
      y = (y * y) % mod;
      exponent = exponent / 2;
   }
   return x % mod;
}

int Fermat(ll p, int iterations) {
   int i;
   if (p == 1) {
      return 0;
   }
   for (i = 0; i < iterations; i++) {
      ll a = rand() % (p - 1) + 1;
```

```
        if (modulo(a, p - 1, p) != 1) {
            return 0;
        }
    }
    return 1;
}
int main() {
    int iteration = 50;
    ll num;
    printf("Mohan Thapa Mashrangi\n");
    printf("Enter integer to test primality: ");
    scanf("%lld", &num);
    if (Fermat(num, iteration) == 1)
        printf("%lld is prime ", num);
    else
        printf("%lld is not prime ", num);
    return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
Enter integer to test primality: 9
9 is not prime %
```

Conclusion:

Hence, in this way we can apply Fermat’s little Theorem in the laboratory.

Task 4:

Title: Write a program to generate a public and private key using RSA algorithm. Also, encrypt a message "CAB College" and again decrypt it using the algorithm.

Theory:

The RSA algorithm is a widely used public-key cryptosystem for secure data transmission. It involves generating a pair of public and private keys, and the security of the algorithm relies on the difficulty of factoring the product of two large prime numbers.

Key Generation:

- Choose two large prime numbers, p and q.
- Compute n = p * q.
- Calculate Euler's totient function, phi(n) = (p-1) * (q-1).
- Choose an integer e such that 1 < e < phi(n) and e is coprime with phi(n).
- Calculate d such that d * e ≡ 1 (mod phi(n)).
- The public key is (n, e), and the private key is (n, d).

Encryption:

- Represent the plaintext message M as an integer.
- Compute the ciphertext C ≡ M^e (mod n) using the public key.

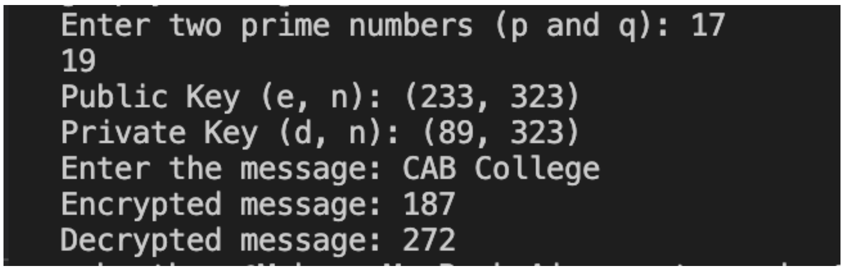
Decryption:

- Compute the original message M ≡ C^d (mod n) using the private key.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
int modulo(int a, int b, int mod) {
    int res = 1;
    while (b > 0) {
        if (b & 1) {
            res = (res * a) % mod;
        }
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
int isPrime(int num) {
    if (num <= 1) {
        return 0;
    }
    for (int i = 2; i * i <= num; ++i) {
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}
int generateRandomPrime() {
    int prime;
    do {
        prime = rand() % 100 + 50;  // Generate a random number between 50 and 150
    } while (!isPrime(prime));
    return prime;
}
int generatePublicKey(int phi) {
    int e;
    do {
        e = rand() % (phi - 2) + 2;  // Generate a random number between 2 and phi-1
    } while (gcd(e, phi) != 1);
    return e;
}
int generatePrivateKey(int e, int phi) {
    int d = 1;
    while ((e * d) % phi != 1) {
        d++;
    }
    return d;
}
int main() {
    srand(time(NULL));
    int p, q, n, phi, e, d, message;
    printf("Enter two prime numbers (p and q): ");
    scanf("%d%d", &p, &q);
    if (!isPrime(p) || !isPrime(q)) {
        printf("Both numbers must be prime.\n");
        return 1;
    }
    n = p * q;
    phi = (p - 1) * (q - 1);
    e = generatePublicKey(phi);
    d = generatePrivateKey(e, phi);
    printf("Public Key (e, n): (%d, %d)\n", e, n);
    printf("Private Key (d, n): (%d, %d)\n", d, n);
    printf("Enter the message: ");
    scanf("%d", &message);
    int cipher = modulo(message, e, n);
    printf("Encrypted message: %d\n", cipher);
    int decrypted = modulo(cipher, d, n);
    printf("Decrypted message: %d\n", decrypted);
    return 0;
}
```

Output:



```
Enter two prime numbers (p and q): 17
19
Public Key (e, n): (233, 323)
Private Key (d, n): (89, 323)
Enter the message: CAB College
Encrypted message: 187
Decrypted message: 272
```

Conclusion:

Hence, in this way we can implement encryption and decryption using RSA algorithm in the laboratory.

Task 5:

Title: Write a program to calculate the Key for two persons using the Diffie Hellman Key exchange algorithm.

Theory:

The Diffie-Hellman algorithm is used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

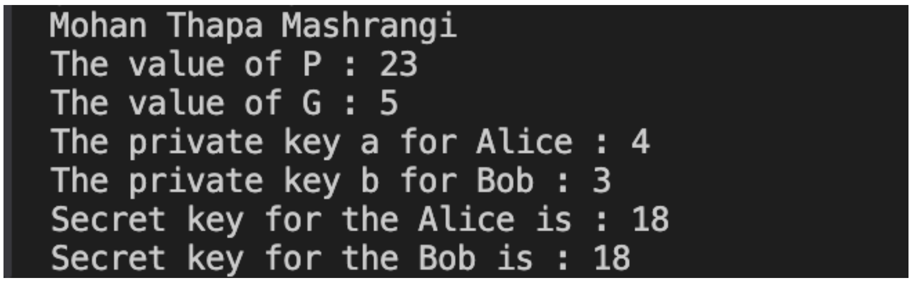
For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b. P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Source Code:

```
#include <cmath>
#include <iostream>
using namespace std;

long long int power(long long int a, long long int b,
                    long long int P)
{
   if (b == 1)
       return a;

   else
       return (((long long int)pow(a, b)) % P);
}

// Driver program
int main()
{
   long long int P, G, x, a, y, b, ka, kb;
```

```
    // Both the persons will be agreed upon the
    // public keys G and P
   cout<<"Mohan Thapa Mashrangi";
    P = 23; // A prime number P is taken
    cout << "The value of P : " << P << endl;
    G = 5; // A primitive root for P, G is taken
    cout << "The value of G : " << G << endl;
    // Alice will choose the private key a
    a = 4; // a is the chosen private key
    cout << "The private key a for Alice : " << a << endl;
    x = power(G, a, P); // gets the generated key
    // Bob will choose the private key b
    b = 3; // b is the chosen private key
    cout << "The private key b for Bob : " << b << endl;
    y = power(G, b, P); // gets the generated key
    // Generating the secret key after the exchange
    // of keys
    ka = power(y, a, P); // Secret key for Alice
    kb = power(x, b, P); // Secret key for Bob
    cout << "Secret key for the Alice is : " << ka << endl;
    cout << "Secret key for the Bob is : " << kb << endl;
    return 0;
}
```

Output:



```
Mohan Thapa Mashrangi
The value of P : 23
The value of G : 5
The private key a for Alice : 4
The private key b for Bob : 3
Secret key for the Alice is : 18
Secret key for the Bob is : 18
```

Conclusion:

Hence, in this way we can observe key generation and sharing using Diffie-Helman Key Sharing Algorithm in the laboratory.

Lab-4

Task 1:

Title: Write a program to implement MD4 and MD5 algorithm using library functions.

Theory:

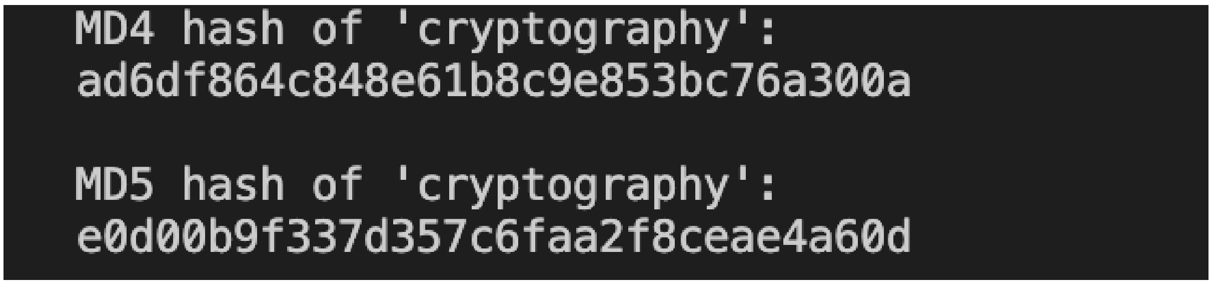
MD4 and MD5 are cryptographic hash functions designed by Rivest. Several hash functions have been influenced by their design. Practical attacks exist for MD4 and MD5, with high impact on commonly used applications.

MD4 and MD5 are the initial members of the MD4 type hash functions. Both were designed by Rivest. They take variable length input messages and hash them to fixed-length outputs. Both operate on 512-bit message blocks divided into 32-bit words and produce a message digest of 128 bits. First, the message is padded according to the so-called Merkle-Damgård strengthening technique. Next, the message is processed block by block by the underlying compression function. This function initialises four 32-bit chaining variables to a fixed value prior to hashing the first message block, and to the current hash value for the following message blocks. Each step of the compression function updates in turn one of the chaining variables according to one message word. Both compression functions are organised into rounds of 16 steps each. MD4 has three such rounds, while MD5 consists of 4 rounds

Source Code:

```
import hashlib
def md4_example(data):
   md4_hash = hashlib.new('md4', data.encode()).hexdigest()
   print(f"MD4 hash of '{data}':\n{md4_hash}\n")
def md5_example(data):
   md5_hash = hashlib.md5(data.encode()).hexdigest()
   print(f"MD5 hash of '{data}':\n{md5_hash}\n")
if __name__ == "__main__":
   data = "cryptography"
   md4_example(data)
   md5_example(data)
```

Output:



```
MD4 hash of 'cryptography':
ad6df864c848e61b8c9e853bc76a300a

MD5 hash of 'cryptography':
e0d00b9f337d357c6faa2f8ceae4a60d
```

Conclusion:

Hence, in this way we can implement use MD4 and MD5 hash algorithm in the laboratory.

Task 2:

Title: Write a program to implement SHA-1 and SHA-2 algorithm using library functions.

Theory:

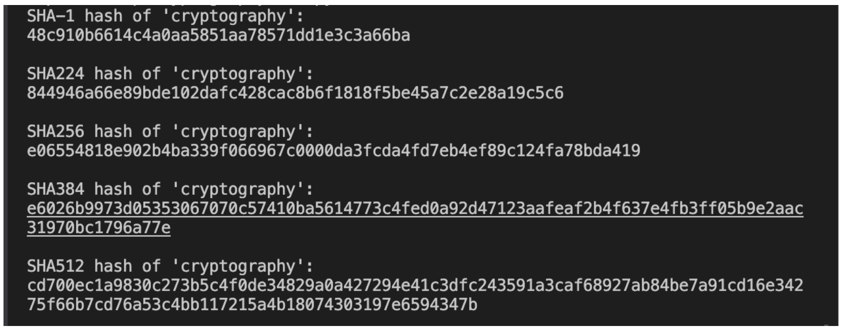
SHA1 is a cryptographic hash function which is designed by United States National Security Agency. It takes an input and produces a 160 bits hash value. Further the output produced by this function is converted into a 40 digits long hexadecimal number. It is a U.S. Federal Information Processing Standard. It was first published in 1995. It is successor to SH0 published in 1993.

SHA1 is also a cryptographic hash function which is designed by United States National Security Agency. It is constructed using the Merkle-Damgard structure from a one-way compression function. The compression function used is constructed using the Davies-Meyer structure from a classified block cipher. It was first published in 2001. It is successor to SH1.

Source Code:

```
import hashlib
def sha1_example(data):
   sha1_hash = hashlib.sha1(data.encode()).hexdigest()
   print(f"SHA-1 hash of '{data}':\n{sha1_hash}\n")
def sha2_example(data, algorithm='sha256'):
   sha2_hash = hashlib.new(algorithm, data.encode()).hexdigest()
   print(f"{algorithm.upper()} hash of '{data}':\n{sha2_hash}\n")
if __name__ == "__main__":
   data = "cryptography"
   sha1_example(data)
   sha2_example(data, algorithm='sha224')
   sha2_example(data, algorithm='sha256')
   sha2_example(data, algorithm='sha384')
   sha2_example(data, algorithm='sha512')
```

Output:



```
SHA-1 hash of 'cryptography':
48c910b6614c4a0aa5851aa78571dd1e3c3a66ba

SHA224 hash of 'cryptography':
844946a66e89bde102dafc428cac8b6f1818f5be45a7c2e28a19c5c6

SHA256 hash of 'cryptography':
e06554818e902b4ba339f066967c0000da3fcda4fd7eb4ef89c124fa78bda419

SHA384 hash of 'cryptography':
e6026b9973d05353067070c57410ba5614773c4fed0a92d47123aafeaf2b4f637e4fb3ff05b9e2aac
31970bc1796a77e

SHA512 hash of 'cryptography':
cd700ec1a9830c273b5c4f0de34829a0a427294e41c3dfc243591a3caf68927ab84be7a91cd16e342
75f66b7cd76a53c4bb117215a4b18074303197e6594347b
```

Conclusion:

Hence, in this way we can implement use SHA1 and SHA2 hash algorithm in the laboratory.

Task 3:

Title: Download SSL (Digital Certificate) of a website and analyze its content.

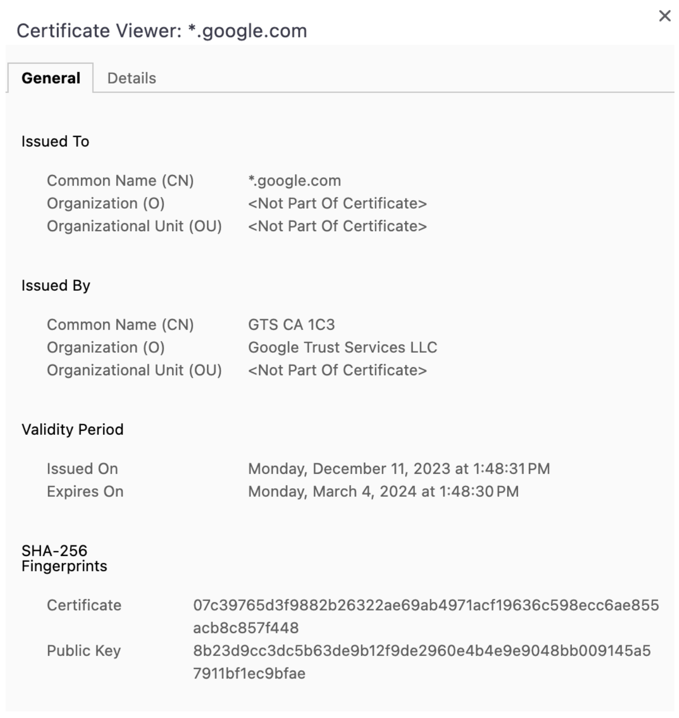
Theory:

An SSL certificate is a Digital certificate that can be used for authentication of a website, and it creates a secure connection between client and web server. When a certificate is installed, it makes the website from HTTP to HTTPS.

Working process of SSL certificate :

The SSL certificate enables the encryption of data which is then sent to the server-side. It has two keys one is public and the other one is a private key. Data encrypted with the public key can be decrypted with a private key only. The web server with a private key can understand the data. If data packets are stolen from in between those are useless because they are encrypted

Certificate:



Certificate Viewer: *.google.com

General | Details

Issued To

Common Name (CN)	*.google.com
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

Common Name (CN)	GTS CA 1C3
Organization (O)	Google Trust Services LLC
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Monday, December 11, 2023 at 1:48:31 PM
Expires On	Monday, March 4, 2024 at 1:48:30 PM

SHA-256 Fingerprints

Certificate	07c39765d3f9882b26322ae69ab4971acf19636c598ecc6ae855 acb8c857f448
Public Key	8b23d9cc3dc5b63de9b12f9de2960e4b4e9e9048bb009145a5 7911bf1ec9bfae

Conclusion:

Hence, in this way we can observe and analyze the SSL of a website.

Task 4:

Title: Write a malicious logic code program that performs some malicious code.

Theory:

Malicious Logic is hardware, firmware, or software that is intentionally included or inserted in a system to perform an unauthorised function or process that will have adverse impact on the confidentiality, integrity, or availability of an information system.

Malware is a software that gets into the system without user consent with an intention to steal private and confidential data of the user that includes bank details and password. They also generates annoying pop up ads and makes changes in system settings

They get into the system through various means:

1. Along with free downloads.
2. Clicking on suspicious link.
3. Opening mails from malicious source.
4. Visiting malicious websites.
5. Not installing an updated version of antivirus in the system.

Types:

1. Virus
2. Worm
3. Logic Bomb
4. Trojan/Backdoor
5. Rootkit
6. Advanced Persistent Threat
7. Spyware and Adware

Source Code:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
        ofstream fout;
        string line = "aaa";
        string txt = ".txt";
        // by default ios::out mode, automatically deletes
        // the content of file. To append the content, open in ios:app
        // fout.open("sample.txt", ios::app)
        for (int i = 0; i <= 10; i++) {
            string name = to_string(i);
            string fname = name + txt;
            // Open the file outside the loop
            fout.open(fname);
            // Execute a loop if the file is successfully opened
           while (fout) {
                for (int j = 0; j <= 10000; j++) {
                    // Generate large text content, you can modify this as needed
                    fout << "This is line " << j << " in file " << i << endl;
                }
                // Write line in the file
                //fout << line << endl;
            fout.close();
            }
        }
    }
```
Output:

On Running the above C code, it generates 10 files with each having 10000 lines of text amounting to each file of around 300 kb. On modifying the number of iterations of the file generation in the outer loop of 'i', the number of files being generated can be modified.

Conclusion:

Hence, in this way we can create and see the effect of a simple malicious logic in the laboratory.