Lab 2

Lab 2.1: Write a program to implement the DES key generation process to generate subkeys. Also, show the subkeys generated at each round.

Algorithm:

STEP 1: Input:

- Accept a 64-bit key as input.

STEP 2: Initial permutation (PC1):

- Apply the PC1 (Permuted Choice 1) table to permute the 64-bit key, rearranging its bits to create a 56-bit key.

STEP 3: Key splitting:

- Divide the 56-bit key into two halves of 28 bits each, C0 and D0.

STEP 4: Round key generation loop:

STEP 5: Repeat the following steps for 16 rounds:

- Left shift:
  i. Shift C0 and D0 left by 1 or 2 bits according to the round number (specified in the DES specification).
- Permutation (PC2):
  i. Combine C0 and D0 back into a 56-bit key.
  ii. Apply the PC2 (Permuted Choice 2) table to select 48 bits from the 56-bit key, forming the subkey for this round.
- Output:
  i. Display or store the generated 48-bit subkey.

STEP 6: Output:

- After 16 rounds, you'll have 16 distinct 48-bit subkeys.

Source Code:

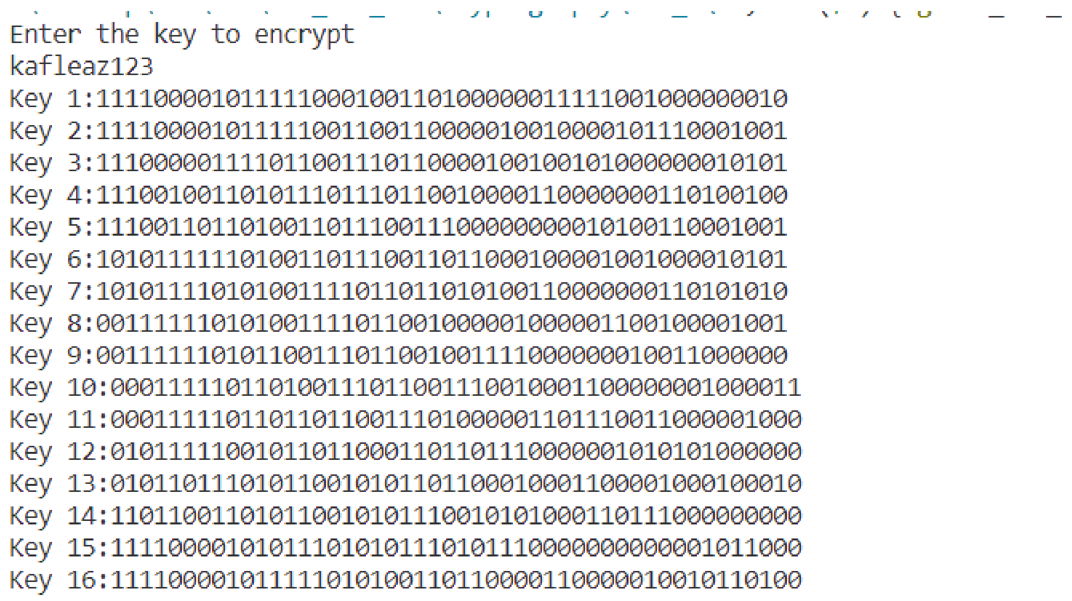
```
#include<iostream>
#include<string>
#include<bitset>

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);
```

}

<u>Output:</u>



```
Enter the key to encrypt
kafleaz123
Key 1:111100001011111000100110100000001111100100000010
Key 2:111100001011111001100110000010010000101110001001
Key 3:111000001111011001110110000100100101000000010101
Key 4:111001001101011101110110010000110000000110100100
Key 5:111001101101001101110011100000000010100110001001
Key 6:101011111101001101110011011000100001001000010101
Key 7:101011110101001111011011010100110000000110101010
Key 8:001111110101001111011001000001000001100100001001
Key 9:001111110101100111011001001111000000010011000000
Key 10:000111110110100111011001110010001100000001000011
Key 11:000111110110110110011101000001101110011000001000
Key 12:010111110010110110001101101110000001010101000000
Key 13:010110111010110010101101100010001100001000100010
Key 14:110110011010110010101110010101000110111000000000
Key 15:111100001010111010101110101110000000000001011000
Key 16:111100001011111010100110110000110000010010110100
```

Lab 2.2 Implement the initial permutation of the DES algorithm by applying the initial permutation to a given plaintext block and observe the result.:

Algorithm:

STEP 1: Define the IP table:

- Create a list or array specifying the new position for each bit in the permuted output.
- The IP table for DES is fixed and can be found in cryptography references.

STEP 2: Input plaintext block:

- Obtain the 64-bit plaintext block to be permuted.

STEP 3: Apply IP:

- Iterate through the 64 bits of the plaintext block:
    1. For each bit, look up its new position in the IP table.
    2. Place the bit at that new position in the output block.

STEP 4: Observe the permuted block:

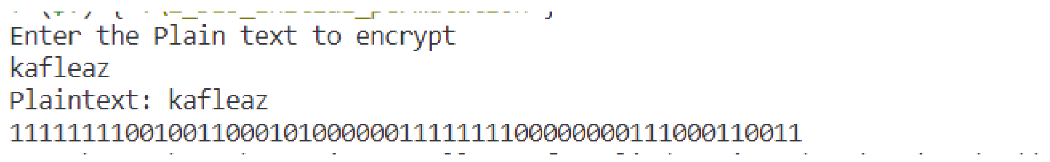
- The resulting 64-bit block is the output of the initial permutation.

Source Code:

```
#include<iostream>
#include<string>
#include<bitset>
using namespace std;
void initial_permutation(string Plain_Text)
{
   //Assuming each character is 1 bit instead of 1 byte
         int initial_perm[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
      };
   string Encrypt;
         for(int i=0;i<64;i++){
            Encrypt += Plain_Text[initial_perm[i]];
         }
         cout << Encrypt;
   cout<<"\n";
}
```

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

int main()
{
    string Plain_Text, Plain_Text_bin;
    cout << "Enter the Plain text to encrypt" << endl;
    cin >> Plain_Text;
    cout<<"Plaintext: " << Plain_Text <<endl;
    Plain_Text_bin = TextToBinaryString(Plain_Text).substr(0, 64);
    initial_permutation(Plain_Text_bin);
}
```

<u>Output:</u>



```
Enter the Plain text to encrypt
kafleaz
Plaintext: kafleaz
111111110010011000101000000011111111000000000111000110011
```

Lab 2.3: Write a program to apply the round function to a given 32-bit data and subkey, and display the intermediate results.

Algorithm:

STEP 1: Expand data: Apply the E-box permutation to the 32-bit data, resulting in a 48-bit expanded block.

STEP 2: XOR with subkey: Perform bitwise XOR between the expanded data and the 48-bit subkey.

STEP 3: Split into blocks: Divide the 48-bit output into 8 blocks of 6 bits each.

STEP 4: Substitute blocks: Apply S-boxes:

- For each 6-bit block:
    - Use the corresponding S-box (S1 to S8) to replace the block with its 4-bit output.
    - Access the S-box using the first and last bits of the 6-bit block as row and column indices.

STEP 5: Permute substituted blocks: Apply the P-box permutation to rearrange the 32 bits resulting from the S-box substitutions.

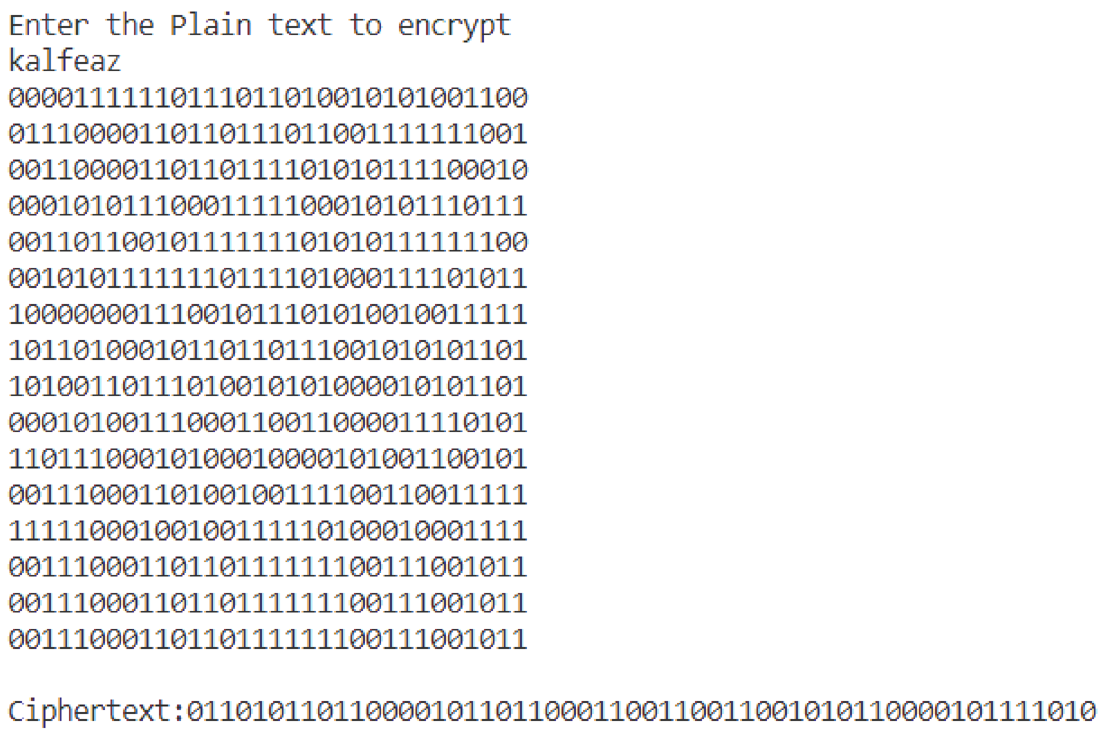
STEP 6: Return output: The final 32-bit output after the P-box is the round function's output (Right Half).

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 << "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];
            }
            //cout << "Round " << i + 1 << " Key: ";
            // 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<<"\n";
        cout <<"Ciphertext:" <<ciphertext << endl;
    }
```

Output:



```
Enter the Plain text to encrypt
kalfeaz
00001111110111011010010101001100
01110000110110111011001111111001
00110000110110111101010111100010
00010101110001111100010101110111
00110110010111111101010111111100
00101011111110111101000111101011
10000000111001011101010010011111
10110100010110110111001010101101
10100110111010010101000010101101
00010100111000110011000011110101
11011100010100010000101001100101
00111000110100100111100110011111
11111000100100111110100010001111
00111000110110111111100111001011
00111000110110111111100111001011
00111000110110111111100111001011

Ciphertext:0110101101100001011011000110011001100101011000010111 1010
```

Lab 2.4: Implement the IDEA key scheduling algorithm to generate subkeys from the main encryption key.

Algorithm:

STEP 1: Input:

- The 128-bit main encryption key (K)

STEP 2: Initialization:

- Divide K into eight 16-bit subkeys: K1, K2, ..., K8
- Create a 128-bit key register (KR) and initialize it with K

STEP 3: Key Scheduling Loop:

- Repeat the following steps eight times:

  Rotate: Rotate the 16 bits in KR left by 25 bits (equivalent to a left circular shift by 9 bits)

STEP 4: Split:

- Divide KR into eight 16-bit subkeys: KR1, KR2, ..., KR8

  STEP 5: Output Subkeys:

- Use KR1, KR2, ..., KR8 as the subkeys for the next eight rounds of IDEA encryption

  STEP 6: Output:

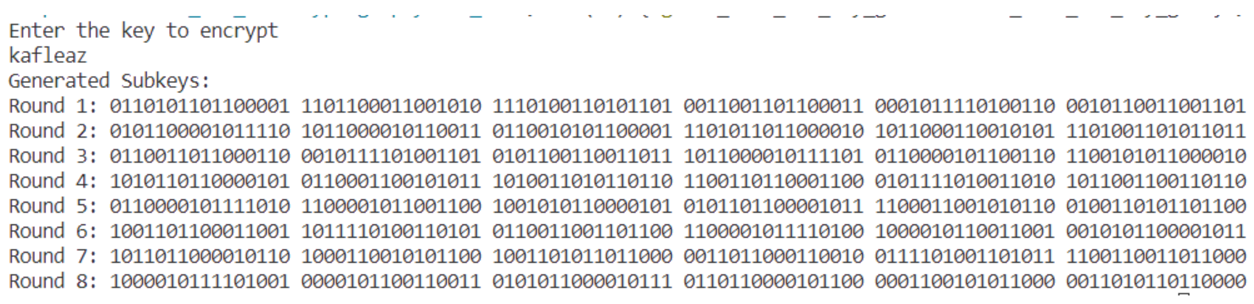
- The 52 subkeys generated from the key scheduling algorithm.

Source Code:

```
#include<iostream>
#include<string>
#include<bitset>
using namespace std;
const int KEY_SIZE = 128;
const int ROUNDS = 8;
// IDEA round keys as strings
string subkeys[ROUNDS][6];
// Function to rotate a string left by n bits
string rotateLeft(const string& str, int n) {
   return str.substr(n) + str.substr(0, n);
}
```

```
// IDEA key scheduling function
void IDEA_key_schedule(string main_key) {
    for (int round = 0; round < ROUNDS; ++round) {
        for (int i = 0; i < 6; ++i) {
            subkeys[round][i] = main_key.substr(0, 16);
            main_key = rotateLeft(main_key, 25);
        }
    }
}
string TextToBinaryString(string words)
{
    string binaryString = "";
    for (char& _char : words) {
        binaryString +=bitset<8>(_char).to_string();
    }
    return binaryString;
}
int main() {

    string main_key, key_bin;
    cout << "Enter the key to encrypt" << endl;
    cin >> main_key;
    key_bin = TextToBinaryString(main_key).substr(0, 128);
    IDEA_key_schedule(key_bin);
    // Output the generated subkeys
    cout << "Generated Subkeys:" << endl;
    for (int round = 0; round < ROUNDS; ++round) {
        cout << "Round " << round + 1 << ": ";
        for (int i = 0; i < 6; ++i) {
            cout << subkeys[round][i] << " ";
        }
        cout << endl;
    }
    return 0;}
```

Output:



```
Enter the key to encrypt
kafleaz
Generated Subkeys:
Round 1: 0110101101100001 1101100011001010 1110100110101101 0011001101100011 0001011110100110 0010110011001101
Round 2: 0101100001011110 1011000010110011 0110010101100001 1101011011000010 1011000110010101 1101001101011011
Round 3: 0110011011000110 0010111101001101 0101100110011011 1011000010111101 0110000101100110 1100101011000010
Round 4: 1010110110000101 0110001100101011 1010011010110110 1100110110001100 0101111010011010 1011001100110110
Round 5: 0110000101111010 1100001011001100 1001010110000101 0101101100001011 1100011001010110 0100110101101100
Round 6: 1001101100011001 1011110100110101 0110011001101100 1100001011110100 1000010110011001 0010101100001011
Round 7: 1011011000010110 1000110010101100 1001101011011000 0011011000110010 0111101001101011 1100110011011000
Round 8: 1000010111101001 0000101100110011 0101011000010111 0110110000101100 0001100101011000 0011010110110000
```

Lab 2.5: 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.

Algorithm:

SubBytes:

STEP 1: Input: A 4x4 state matrix representing the current state of the data.

STEP 2: Process each byte:

a. Iterate through each byte in the state matrix.
b. Use the S-box (a fixed 16x16 lookup table) to substitute the current byte with the corresponding value in the S-box.

ShiftRows:

STEP 1: Shift rows cyclically:

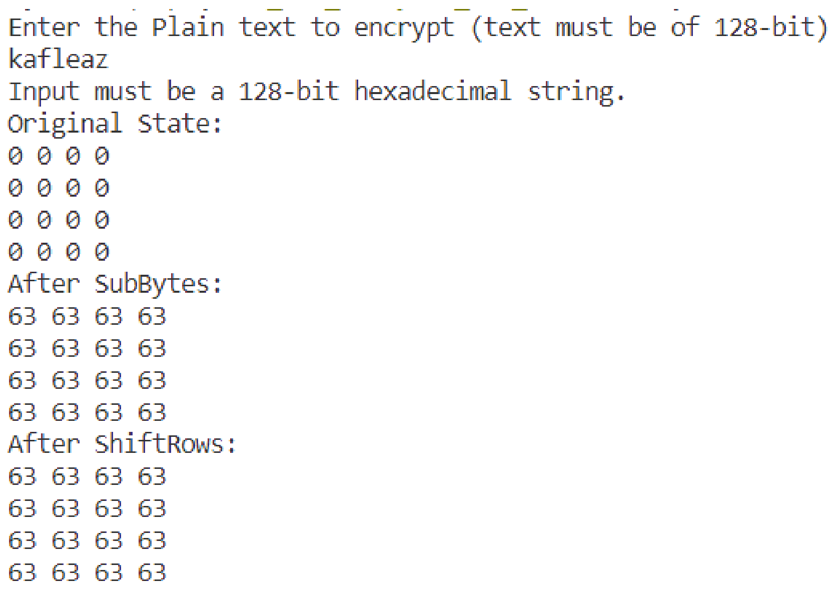
a. Row 1: Keep it unchanged.
b. Row 2: Shift all bytes one position to the left.
c. Row 3: Shift all bytes two positions to the left.
d. Row 4: Shift all bytes three positions to the left.

Source Code:

```
#include<iostream>
#include<vector>
#include<string>
#include<bitset>
using namespace std;
// S-box for SubBytes operation (hexadecimal representation)
const unsigned char sBox[256] = {
   // S-box values (in hexadecimal)
   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
};
// Shift positions for ShiftRows operation
const int shiftPositions[4][4] = {
    {0, 1, 2, 3},
    {1, 2, 3, 0},
    {2, 3, 0, 1},
    {3, 0, 1, 2}
};

// Function to apply SubBytes operation
```

```
void SubBytes(vector<vector<uint8_t>>& state) {
    for (int row = 0; row < 4; row++) {
        for (int col = 0; col < 4; col++) {
            state[row][col] = sBox[state[row][col]];
        }
    }
}
// Function to apply ShiftRows operation
void ShiftRows(vector<vector<uint8_t>>& state) {
    vector<vector<uint8_t>> tempState = state;
    for (int row = 0; row < 4; row++) {
        for (int col = 0; col < 4; col++) {
            state[row][col] = tempState[row][shiftPositions[row][col]];
        }
    }
}
// Function to convert a 128-bit hexadecimal string to a 4x4 state matrix
vector<vector<uint8_t>> HexStringToStateMatrix(const string& hexString) {
    vector<vector<uint8_t>> state(4, vector<uint8_t>(4, 0));
    if (hexString.length() < 32) {
        cout << "Input must be a 128-bit hexadecimal string." << endl;
        return state;
    }
    for (int i = 0; i < 32; i += 2) {
        string byteStr = hexString.substr(i, 2);
        state[i / 8][i % 8 / 2] = stoul(byteStr, nullptr, 16);
    }
    return state;
}
// Function to display the state matrix
void DisplayState(const vector<vector<uint8_t>>& state) {
    for (int row = 0; row < 4; row++) {
        for (int col = 0; col < 4; col++) {
            cout << hex << static_cast<int>(state[row][col]) << " ";
        }
        cout << endl;
    }
}
string BinaryToHex(string binaryStr) {
    string hexStr = "";
    int numBits = binaryStr.size();
    // Ensure the binary string length is a multiple of 4 (for easy conversion to hex)
    int padding = (4 - numBits % 4) % 4;
    string paddedBinaryStr = string(padding, '0') + binaryStr;
    for (int i = 0; i < numBits; i += 4) {
        string nibble = paddedBinaryStr.substr(i, 4);
        int decimalValue = stoi(nibble, nullptr, 2);
        hexStr += "0123456789ABCDEF"[decimalValue];
    }
    return hexStr;
}
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, hex_plainText;
    cout << "Enter the Plain text to encrypt (text must be of 128-bit)" << endl;
    cin >> Plain_Text;
    binary_plaintext = TextToBinaryString(Plain_Text).substr(0, 128);
    hex_plainText = BinaryToHex(binary_plaintext);
    vector<vector<uint8_t>> state = HexStringToStateMatrix(hex_plainText);
    if (state.empty()) {
        return 1;
    }
    cout << "Original State:" << endl;
    DisplayState(state);
    cout << "After SubBytes:" << endl;
    SubBytes(state);
    DisplayState(state);
    cout << "After ShiftRows:" << endl;
    ShiftRows(state);
    DisplayState(state);
    return 0;
}
```

<u>Output:</u>



```
Enter the Plain text to encrypt (text must be of 128-bit)
kafleaz
Input must be a 128-bit hexadecimal string.
Original State:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
After SubBytes:
63 63 63 63
63 63 63 63
63 63 63 63
63 63 63 63
After ShiftRows:
63 63 63 63
63 63 63 63
63 63 63 63
63 63 63 63
```

Lab 2.6: 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.

Algorithm:

STEP 1: Input: A 4x4 state matrix representing the current state of the data.

STEP 2: Process each column:

i. For each column in the state matrix:

   a. Treat the column as a 4-term polynomial over the finite field GF(2^8).

   b. Multiply the polynomial by a fixed 4x4 matrix (defined in AES specifications) using special finite field multiplication rules.

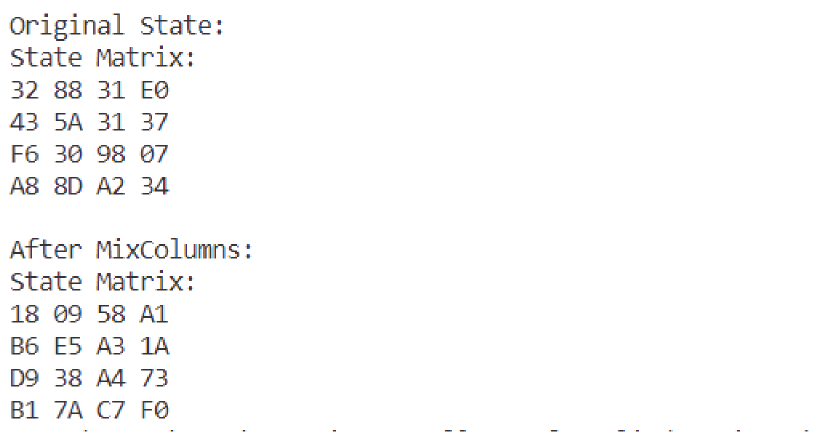
   c. Replace the original column with the resulting column.

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;
    }
}
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("Original State:\n");
    displayState(state);
    mixColumns(state);
    printf("\nAfter MixColumns:\n");
    displayState(state);
    return 0;
}
```

Output:



```
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
```

Lab 3

Lab 3.1: Write a program to implement the Miller-Rabin primality test. Test it with various values of 'n'.

Algorithm:

```
bool isPrime(int n, int k)
STEP 1: Handle base cases for n < 3
STEP 2: If n is even, return false.
STEP 3: Find an odd number d such that n-1 can be written as d*2r.  Note that since n
is odd, (n-1) must be even and r must be greater than 0.
STEP 4: Do following k times
    • if (millerTest(n, d) == false)
    • return false
STEP 5: Return true.
```

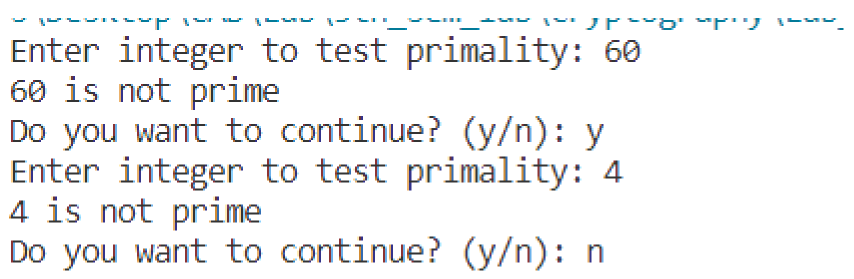
Source Code:

```
#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()
{
  // generateHeader("Program to implement Miller Rabin primality test");
  do
  {
    int iteration = 10;
    long long num;
    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);
    cin.get();
    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:



```
Enter integer to test primality: 60
60 is not prime
Do you want to continue? (y/n): y
Enter integer to test primality: 4
4 is not prime
Do you want to continue? (y/n): n
```

Lab 3.2: Calculate φ(n) (Euler's Totient Function) for a given positive integer 'n.' Verify its correctness for multiple values of 'n.

Algorithm:

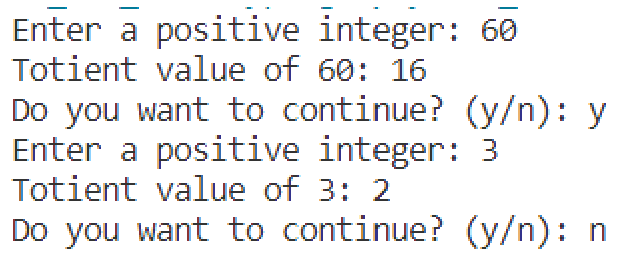
```
STEP 1: Set result to n.
STEP 2: While n is divisible by 2:
Update result by subtracting result // 2.
Divide n by 2.
STEP 3: Start with p = 3.
While p * p is less than or equal to n:
If n is divisible by p:
Update result by subtracting result // p.
While n is divisible by p, divide n by p.
Increment p by 2 (skip even numbers).
STEP 4: If n is greater than 1:
Update result by subtracting result // n.
STEP 5: The final value of result is the Euler Totient Function φ(n).
```

Source Code:

```
#include <iostream>
using namespace std;
void computeTotient(int);
int main()
{
   // generateHeader("Program to compute Totient value.");
   int n;
   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; // indicates not evaluated yet and initializes for product formula.
    for (int p = 2; p <= n; p++)
    {
        // If phi[p] is not computed already, then number p is prime
        if (phi[p] == p)
        {
            phi[p] = p - 1;
            for (int i = 2 * p; i <= n; i += p)
            {
                // Add contribution of p to its multiple i by multiplying with (1 - 1/p)
                phi[i] = (phi[i] / p) * (p - 1);
            }
        }
    }
    cout << "Totient value of " << n << ": " << phi[n] << endl;
}
```

Output:



```
Enter a positive integer: 60
Totient value of 60: 16
Do you want to continue? (y/n): y
Enter a positive integer: 3
Totient value of 3: 2
Do you want to continue? (y/n): n
```

Lab 3.3: Write a program to apply Fermat's Little Theorem to check if a given number, is a probable prime.

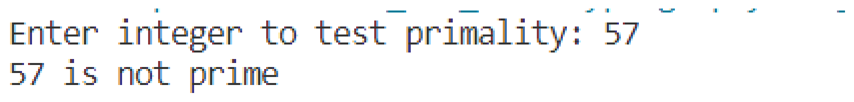
Algorithm:

```
STEP 1: Given integers a and p where p is a prime number.
STEP 2: Calculate exponent as p−2.
STEP 3: Calculate inverse as $a^{exponent}$ (mod p).
STEP 4: The value of inverse is the modular inverse of a modulo 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("Enter integer to test primality: ");
    scanf("%lld", &num);
    if (Fermat(num, iteration) == 1)
        printf("%lld is probably prime ", num);
    else
        printf("%lld is not prime ", num);
    return 0;
}
```

Output:



```
Enter integer to test primality: 57
57 is not prime
```

Lab 3.4: 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.

Algorithm:

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute n as the product of p and q.

STEP-3: Compute (p-1)*(q-1) and store it in z.

STEP-4: Select a random prime number e that is less than that of z.

STEP-5: Compute the private key, d as e * mod^{-1}(z).

STEP-6: The cipher text is computed as message^{e} * mod n.

STEP-7: Decryption is done as cipher^{d} mod n.

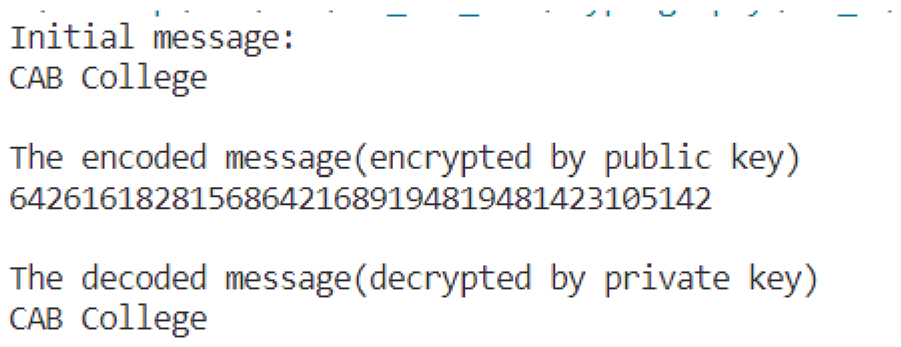
Source Code:

```
#include <bits/stdc++.h>
using namespace std;
set<int>        prime;
int public_key;
int private_key;
int n;
void primefiller()
{
        vector<bool> seive(250, true);
        seive[0] = false;
        seive[1] = false;
        for (int i = 2; i < 250; i++) {
                for (int j = i * 2; j < 250; j += i) {
                        seive[j] = false;
                }
        }
        for (int i = 0; i < seive.size(); i++) {
                if (seive[i])
                        prime.insert(i);
        }
}
```

```
int pickrandomprime()
{
	int k = rand() % prime.size();
	auto it = prime.begin();
	while (k--)
		it++;
	int ret = *it;
	prime.erase(it);
	return ret;
}
void setkeys()
{
	int prime1 = pickrandomprime();
	int prime2 = pickrandomprime();
	n = prime1 * prime2;
	int fi = (prime1 - 1) * (prime2 - 1);
	int e = 2;
	while (1) {
		if (__gcd(e, fi) == 1)
			break;
		e++;
	}
	public_key = e;
	int d = 2;
	while (1) {
		if ((d * e) % fi == 1)
			break;
		d++;
	}
	private_key = d;
}
long long int encrypt(double message)
{
	int e = public_key;
	long long int encrpyted_text = 1;
	while (e--) {
		encrpyted_text *= message;
		encrpyted_text %= n;
	}
	return encrpyted_text;
}
long long int decrypt(int encrpyted_text)
{
	int d = private_key;
	long long int decrypted = 1;
	while (d--) {
		decrypted *= encrpyted_text;
		decrypted %= n;
	}
	return decrypted;
}
vector<int> encoder(string message)
{
	vector<int> form;
	for (auto& letter : message)
		form.push_back(encrypt((int)letter));
	return form;
}
string decoder(vector<int> encoded)
{
	string s;
	for (auto& num : encoded)
		s += decrypt(num);
	return s;
}
int main()
{
	primefiller();
```

```
        setkeys();
        string message = "CAB College";

        vector<int> coded = encoder(message);
        cout << "Initial message:\n" << message;
        cout << "\n\nThe encoded message(encrypted by public "
                "key)\n";
        for (auto& p : coded)
            cout << p;
        cout << "\n\nThe decoded message(decrypted by private "
                "key)\n";
        cout << decoder(coded) << endl;
        return 0;
}
```

<u>Output:</u>



```
Initial message:
CAB College

The encoded message(encrypted by public key)
642616182815686421689194819481423105142

The decoded message(decrypted by private key)
CAB College
```

Lab 3.5: Write a program to calculate the Key for two persons using the Diffie Hellman Key exchange

Algorithm:

STEP-1: Both Alice and Bob shares the same public keys g and p.

STEP-2: Alice selects a random public key a.

STEP-3: Alice computes his secret key A as g^a mod p.

STEP-4: Then Alice sends A to Bob.

STEP-5: Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

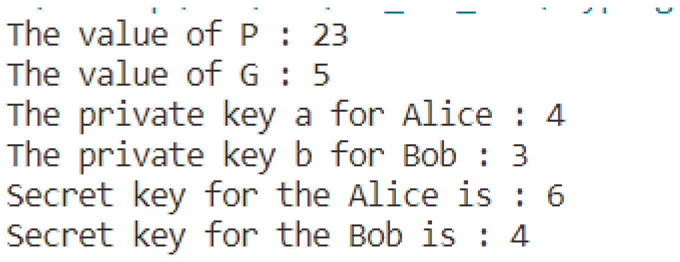
STEP-6: Now both of them compute their common secret key as the other one's secret key power of a mod p

Source Code:

```
#include <cmath>
#include <iostream>
using namespace std;
// Power function to return value of a ^ b mod P
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
        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
        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:

// This code is contributed by Pranay Arora



```
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 : 6
Secret key for the Bob is : 4
```

Lab 4

Lab 4.1: Write a program to implement MD4 and MD5 algorithm using library functions.

MD5 Algorithm:

STEP 1: Read the 128-bit plain text.

STEP 2: Divide into four blocks of 32

STEP 3: Compute the functions f, g, h and i with operations such as, rotations, permutations, etc,.

STEP 4: The output of these functions are combined together as F and performed circular shifting and then given to key round.

STEP 5: Finally, right shift of ‘s’ times are performed and the results are combined together to produce the final output.

MD4 Algorithm:

STEP 1: Set H to initial constant values.

STEP 2: For each 32-bit word in the message:

- Update H using a simple mixing function.

STEP 3: If there are more words in the message, go back to Step 2.

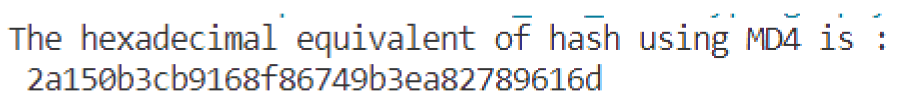
STEP 4: The final hash is H.

STEP 5: Output the final 128-bit hash value.

MD4 Source Code:

```
import hashlib
text = 'Hello There'
hashObject = hashlib.new('md4', text.encode('utf-8'))
digest = hashObject.hexdigest()
print("The hexadecimal equivalent of hash using MD4 is \n: ", end ="")
print(digest)
```

Output:

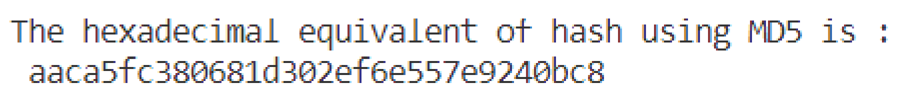


```
The hexadecimal equivalent of hash using MD4 is :
 2a150b3cb9168f86749b3ea82789616d
```

MD5 Source Code:

```
import hashlib
str2hash = "Master Kenobi"
result = hashlib.md5(str2hash.encode())
print("The hexadecimal equivalent of hash using MD5 is  \n: ", end ="")
print(result.hexdigest())
```

Output:



```
The hexadecimal equivalent of hash using MD5 is :
 aaca5fc380681d302ef6e557e9240bc8
```

Lab 4.2: Write a program to implement SHA-1 and SHA-2 algorithm using library functions.

SHA-1 Algorithm:

STEP 1: Read the 256-bit key values.

STEP 2: Divide into five equal-sized blocks named A, B, C, D and E.

STEP 3: The blocks B, C and D are passed to the function F.

STEP 4: The resultant value is permuted with block E.

STEP 5: The block A is shifted right by ‘s’ times and permuted with the result of step-4.

STEP 6: Then it is permuted with a weight value and then with some other key pair and taken as the first block.

STEP 7: Block A is taken as the second block and the block B is shifted by ‘s’ times and taken as the third block.

STEP 8: The blocks C and D are taken as the block D and E for the final output.

SHA-2 Algorithm:

STEP 1: Set the initial hash values (H0 to H7).

STEP 2: Append the message with padding to achieve a length that is a multiple of the block size.

STEP 3: Divide the padded message into blocks.

STEP 4: For each Block

a. Prepare the message schedule (W[0] to W[63]).

b. Initialize working variables (a to h) with the current hash values.

STEP 5: For t = 0 to 63:

Perform logical and bitwise operations based on the SHA-2 specifications.

STEP 6: Update the hash values (a to h) based on the results of the Main Loop.

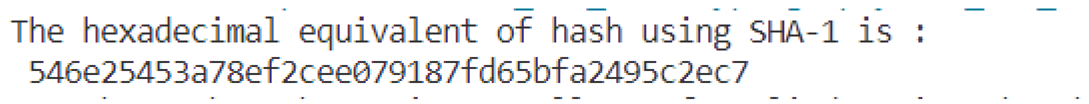
STEP 7: If there are more blocks, go back to Step 4.

STEP 8: Concatenate the final hash values (H0 to H7) to get the SHA-2 hash.

SHA-1 Source Code:

```
import hashlib
text = 'Hello There'
result = hashlib.sha1(text.encode())
print("The hexadecimal equivalent of hash using SHA-1 is : ", end ="")
print(result.hexdigest())
```

SHA-1 Output:

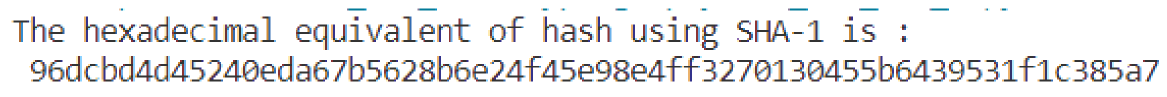


```
The hexadecimal equivalent of hash using SHA-1 is :
 546e25453a78ef2cee079187fd65bfa2495c2ec7
```

SHA-2 Source Code:

```
import hashlib
text = 'Master Kenobi'
result = hashlib.sha256(text.encode())
print("The hexadecimal equivalent of hash using SHA-1 is : ", end ="")
print(result.hexdigest())
```

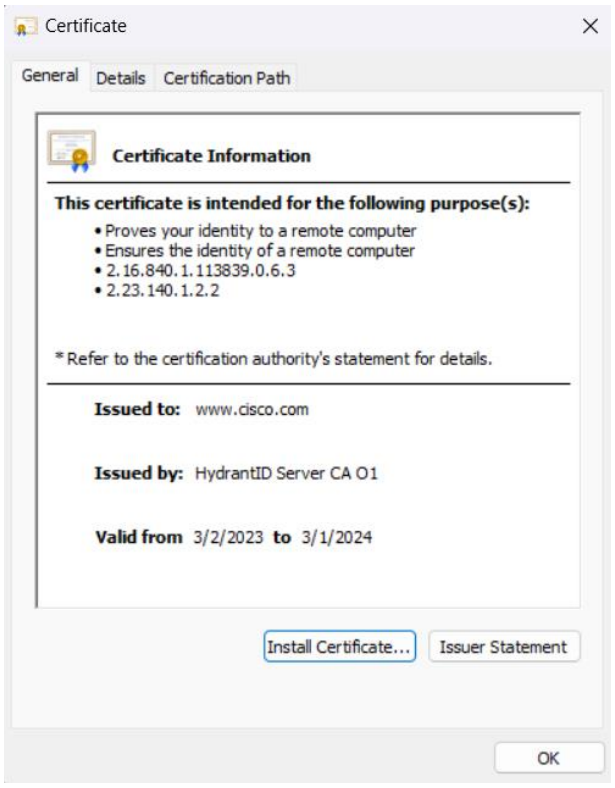
SHA-2 Output:



```
The hexadecimal equivalent of hash using SHA-1 is :
 96dcbd4d45240eda67b5628b6e24f45e98e4ff3270130455b6439531f1c385a7
```

Lab 4.3: Download SSL (Digital Certificate) of a website and analyze its content.

www.cicso.com



Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

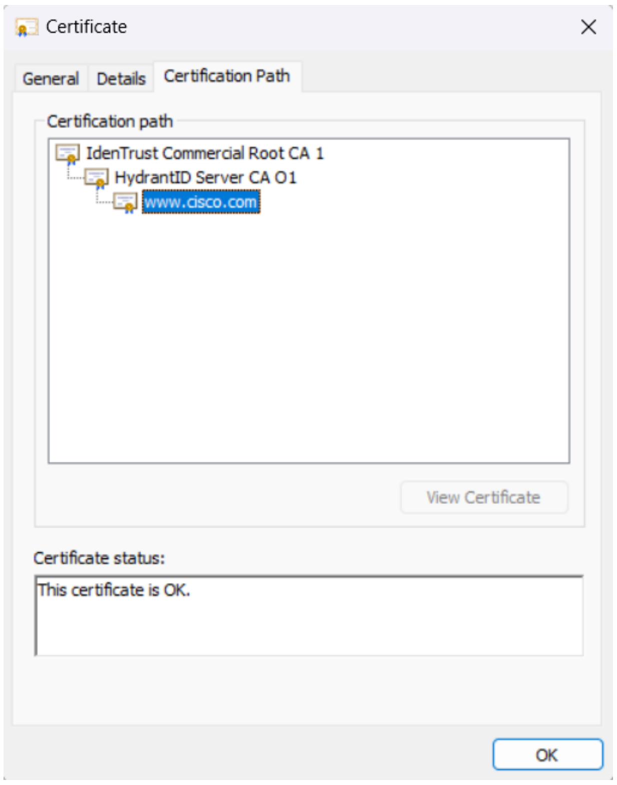
- Proves your identity to a remote computer
- Ensures the identity of a remote computer
- 2.16.840.1.113839.0.6.3
- 2.23.140.1.2.2

* Refer to the certification authority's statement for details.

Issued to: www.cisco.com

Issued by: HydrantID Server CA O1

Valid from 3/2/2023 **to** 3/1/2024



Certificate

General | Details | Certification Path

Certification path

```
IdenTrust Commercial Root CA 1
  HydrantID Server CA O1
    www.cisco.com
```

Certificate status:

This certificate is OK.

General Information:

Issuer: IdenTrust Technologies Corporation (issued by DigiCert Inc)

Issued by: HydrantID Server CA O1

Validity: March 1, 2023 - February 29, 2024 (expire date)

Key Algorithm: RSA (2048 bits)

Signature Algorithm: SHA-256

Certificate type: Domain Validation (DV)

Extended Validation (EV): No

Certificate Transparency (CT): Not included

Serial Number: 4001869eecc535abd516d1156f788630

Detailed Contents:

Validity period: The certificate was valid for one year.

Subject Alternative Names (SANs): This certificate protects not only www.cisco.com but also several other Cisco subdomains, like

- DNS Name=www.cisco.com
- DNS Name=www.static-cisco.com
- DNS Name=www-cloud-cdn.cisco.com
- DNS Name=www.mediafiles-cisco.com
- DNS Name=www-dev-cloud-cdn.cisco.com
- DNS Name=www-stage-cloud-cdn.cisco.com

Extended Validation (EV) and Certificate Transparency (CT) absence: These features could enhance user trust and transparency by validating Cisco's identity more rigorously and publicly recording certificate issuance and revocation.

Signature details: The SHA-256 signature algorithm is currently considered secure.

- 6a9d19a1937a51126d821a9dc8ff80079c452021

Issuer information: IdenTrust is a reputable CA, though issued by DigiCert, a larger and more widely recognized CA.

Organizational information: The certificate includes Cisco's legal name, address, and other organizational details, providing additional identity verification.

Public key:

```
30 82 01 0a 02 82 01 01 00 d8 82 8d 72 62 f1 d6 18 63 06 32 0a bf 9a e4 98 91
72 cb c6 10 fd d9 95 69 60 3e 59 0f 1c 23 04 d8 94 c1 03 b7 ee 18 f2 ef 2d 68
be 6c a4 c3 d0 80 9d 7b ec 60 f9 a8 7f ba 2f 1f 4f 2a cc 57 8f 91 72 bd 93 6c e0
8c e1 05 b0 25 64 1c 9f 0c 19 49 f4 a3 da 78 46 07 44 8c 80 21 49 4c 20 87 57
28 ac a7 e6 e2 8e 49 84 e2 c4 05 13 81 cf ab cf 92 a0 8a 72 c0 4c 15 ee fd a0 0d
be 4c 3e 5b f1 ac 5c 10 e5 5b 59 c6 22 18 03 31 f6 91 7b 60 a6 72 85 20 87 b7
be 5d ec 41 86 69 05 80 c0 fe 9c 63 0d c8 0a 7b e4 e1 ee 5b 00 9b c9 c2 71 30
b9 4e 26 f3 3f 85 a2 44 91 ad a5 21 21 6e 31 66 2a be 9a c5 a7 7f bf 40 54 cc
69 83 80 98 0f 4e 98 15 7e d5 49 9e c8 b8 7f 65 eb c1 48 49 c2 8b 6a 93 2e f1
75 f8 01 5a 98 d0 15 81 7c 18 13 6f 95 c9 67 16 67 a0 9e 88 72 a2 2c c8 33 e1
d4 cf 17 02 03 01 00 01
```

Lab 4.2: Write a malicious logic code program that performs some malicious code.

<u>Theory</u>:

Malicious Logic is hardware, firmware, or software that is intentionally included or inserted in a system to perform an unauthorized 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.

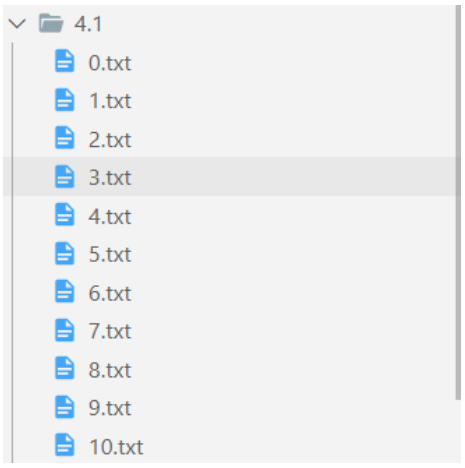
Types:

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

<u>Source Code:</u>

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

int main()
{
            ofstream fout;
            string line = "aaa";
    string txt = ".txt";
    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 <= 10; 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:



```
4.1
    0.txt
    1.txt
    2.txt
    3.txt
    4.txt
    5.txt
    6.txt
    7.txt
    8.txt
    9.txt
    10.txt
```