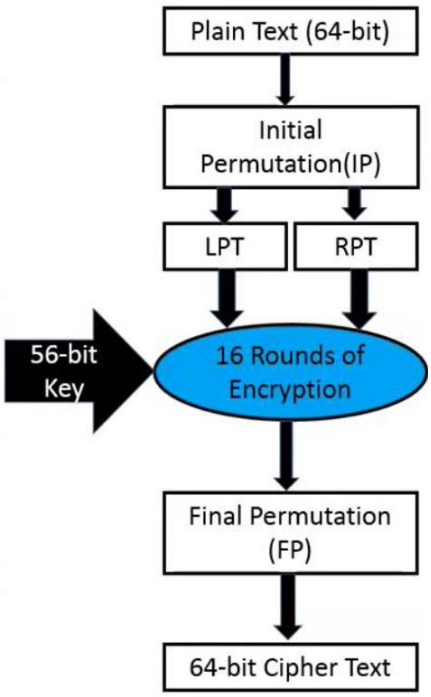
Lab-2

Task 1:

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

Theory:

Data Encryption Standard (DES) is a block cipher with a 56-bit key length that has played a significant role in data security. Data encryption standard (DES) has been found vulnerable to very powerful attacks therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.



Initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

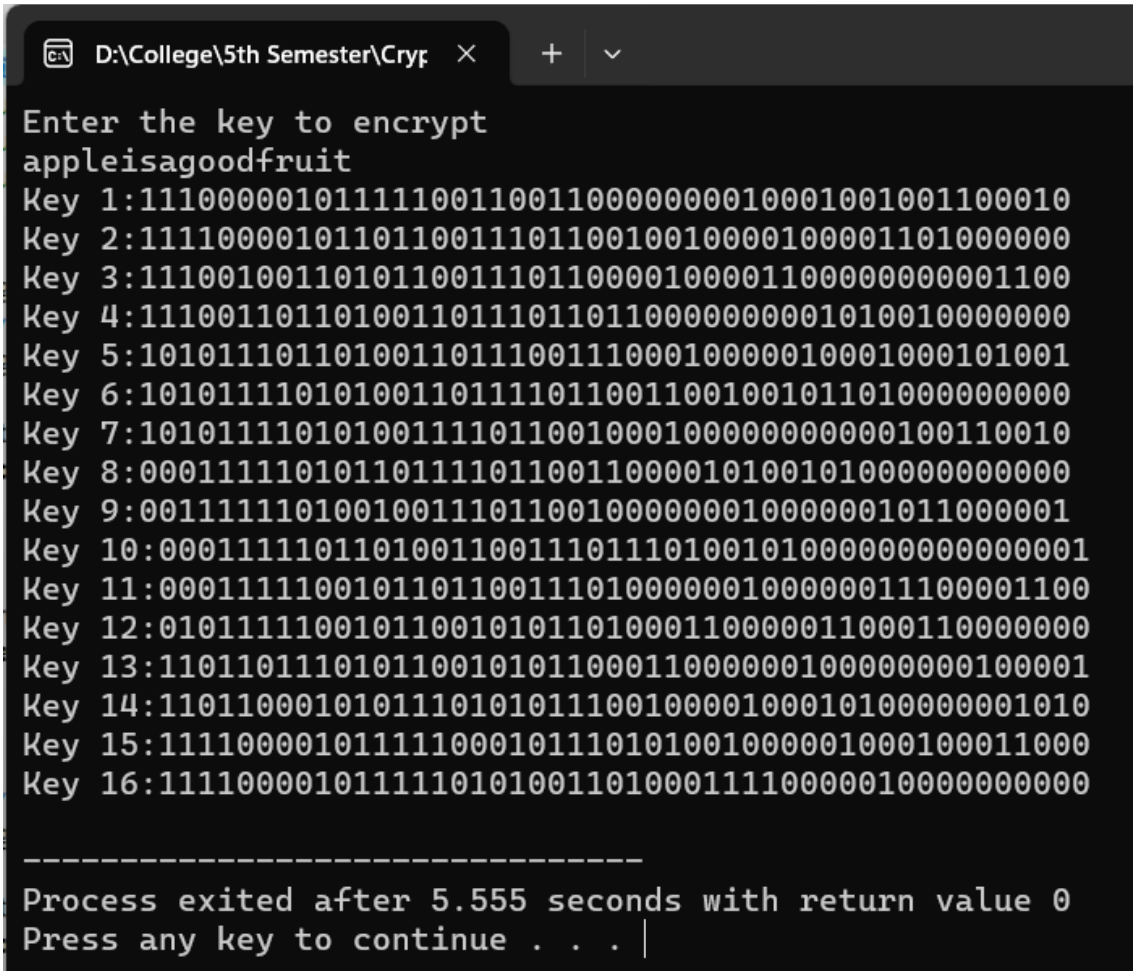
Source Code:

```
#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:



Conclusion:

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

Task 2:

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

Theory:

In block ciphers, including DES, the core encryption process involves multiple rounds. Each round consists of a round function that operates on a portion of the data (usually a block) and a subkey derived from the original key. The round function typically includes operations like substitution, permutation, and bitwise operations to introduce confusion and diffusion in the encryption process.

In the context of DES:

1. DES employs a Feistel network structure, where the data is split into two halves, and operations are applied independently to each half in every round.
2. Each round includes a combination of substitution, permutation, and XOR operations.
3. The key schedule generates subkeys for each round based on the original key, and these subkeys are used in the XOR operations.

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;
    }
```

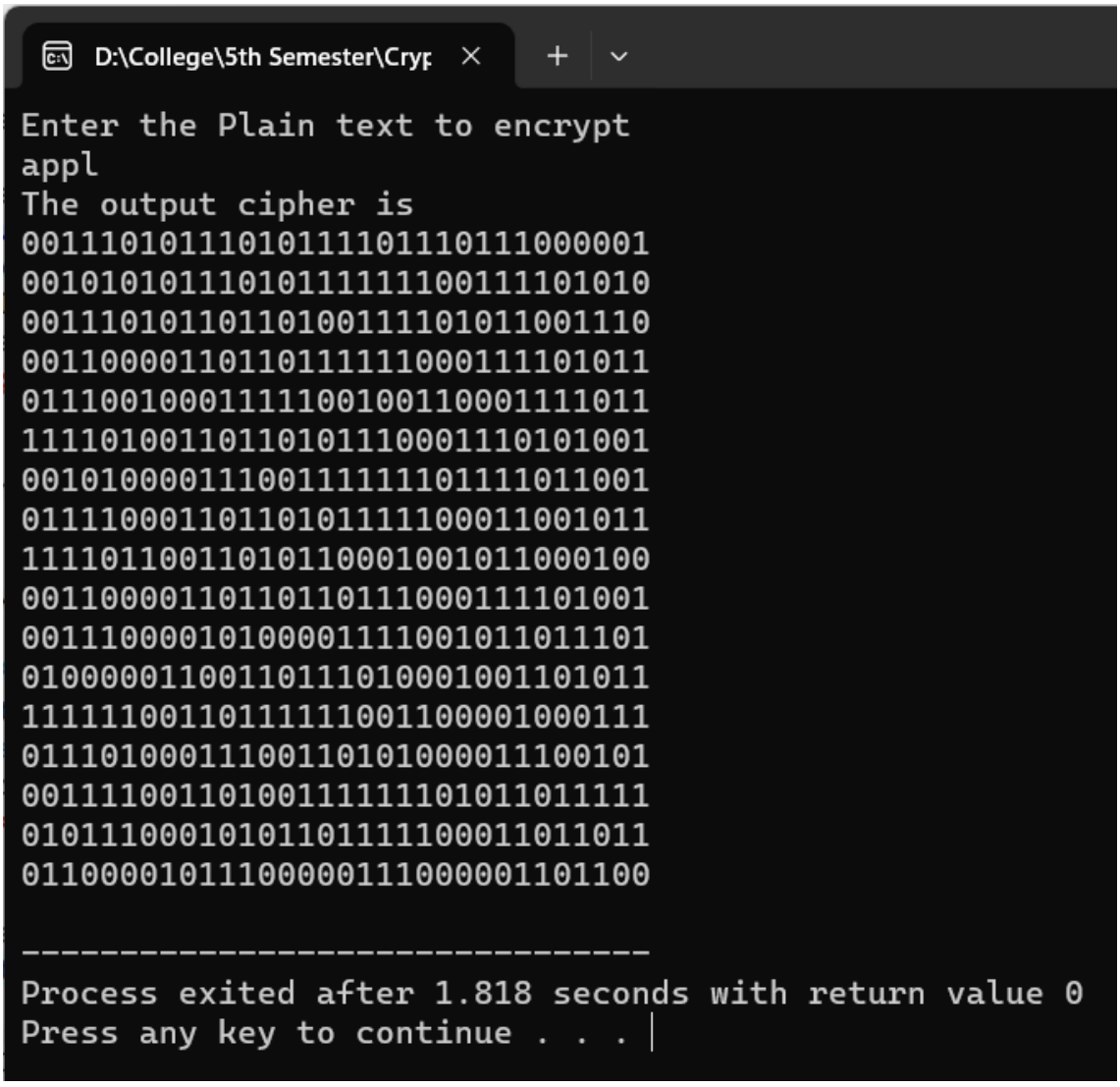
```
    while(binary.length() < 4){
        binary = "0" + binary;
    }
    return binary;
}
int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    int counter = 0;
    int size = binary.length();
    for(int i = size-1; i >= 0; i--)
    {
        if(binary[i] == '1'){
            decimal += 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];
        }
        // 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:



Conclusion:

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

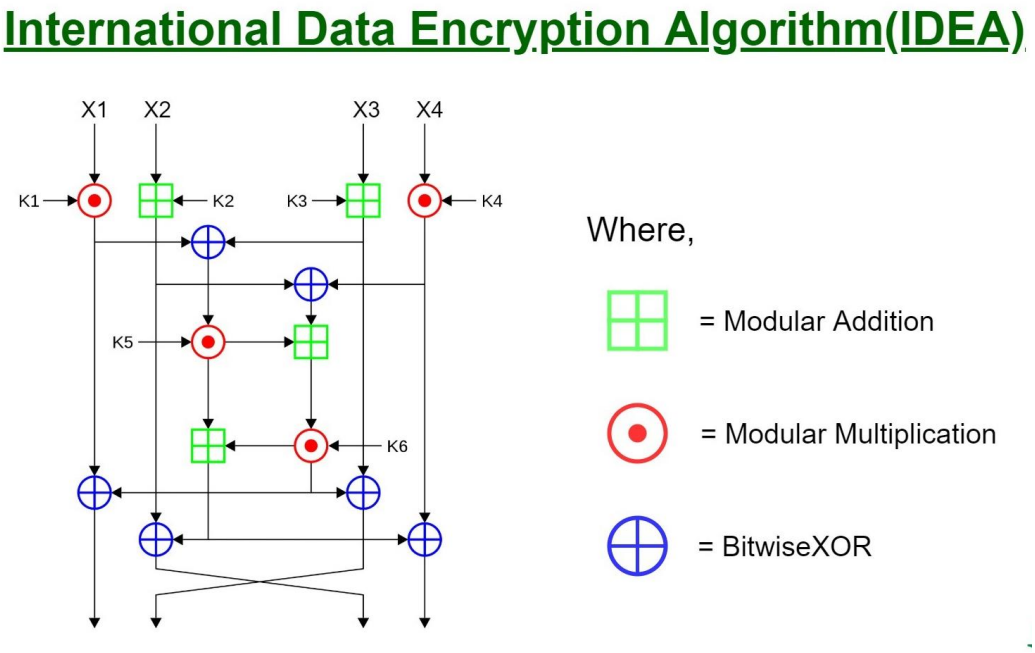
Task 3:

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

Theory:

International Data Encryption Algorithm (IDEA) is a symmetric-key block cipher that was first introduced in 1991. It was designed to provide secure encryption for digital data and is used in a variety of applications, such as secure communications, financial transactions, and electronic voting systems.

IDEA uses a block cipher with a block size of 64 bits and a key size of 128 bits. It uses a series of mathematical operations, including modular arithmetic, bit shifting, and exclusive OR (XOR) operations, to transform the plaintext into ciphertext. The cipher is designed to be highly secure and resistant to various types of attacks, including differential and linear cryptanalysis. IDEA has been widely used in various encryption applications, although it has been largely replaced by newer encryption algorithms such as AES (Advanced Encryption Standard) in recent years. However, IDEA is still considered to be a highly secure and effective encryption algorithm, and it continues to be used in some legacy systems and applications.

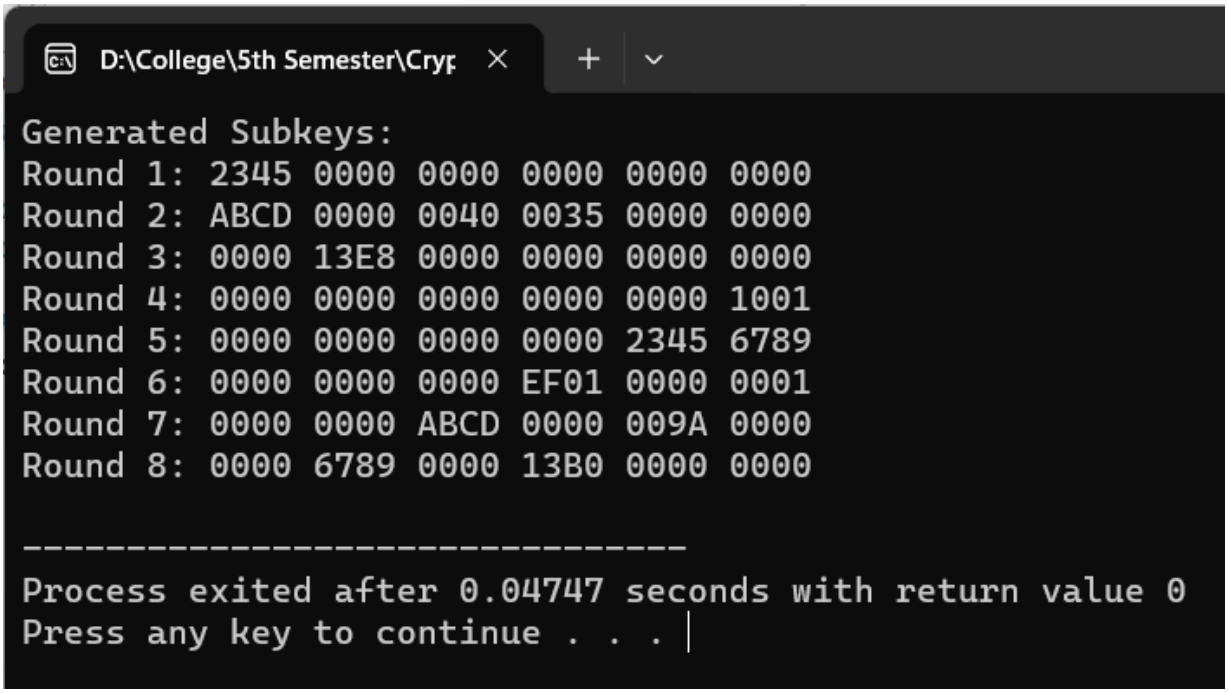


6 subkeys of 4 bits out of the 8 subkeys are used in each complete round, while 4 are used in the half-round. So, 4.5 rounds require 28 subkeys. The given key, 'K', directly gives the first 8 subkeys. By rotating the main key left by 6 bits between each group of 8, further groups of 8 subkeys are created, implying less than one rotation per round for the key (3 rotations).

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);

    // 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:



Conclusion:

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

Task 4:

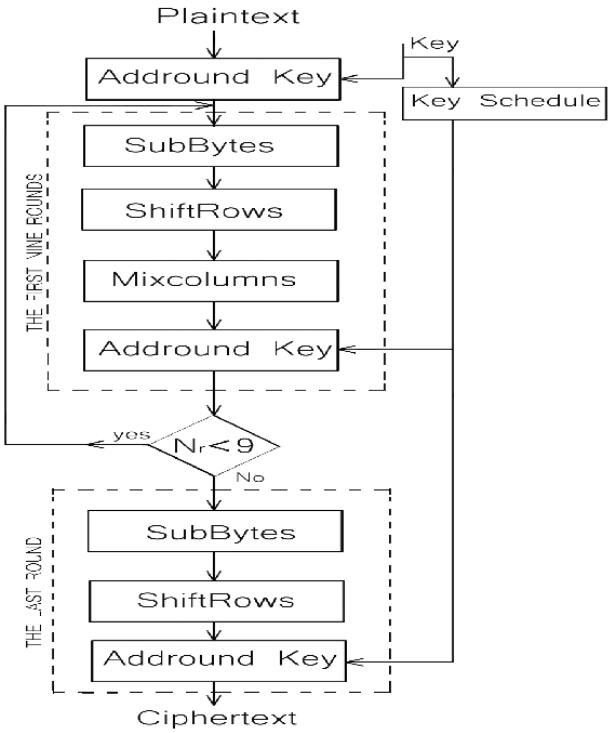
Write a program to implement the AES SubBytes and ShiftRows operations for encryption. Apply these operations to a given state matrix and show the results.

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.

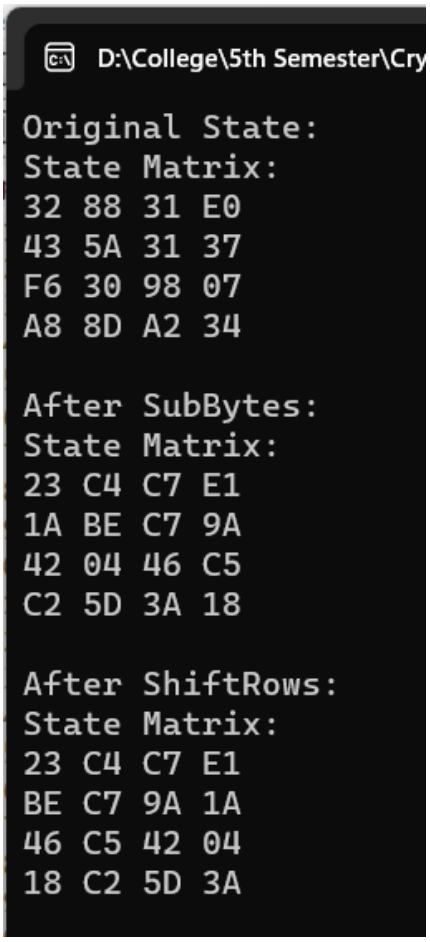
Source Code:

```
#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("Original State:\n");
    displayState(state);

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

    shiftRows(state);
    printf("\nAfter ShiftRows:\n");
    displayState(state);

    return 0;
}
```

Output:



Conclusion:

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

Task 5:

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:

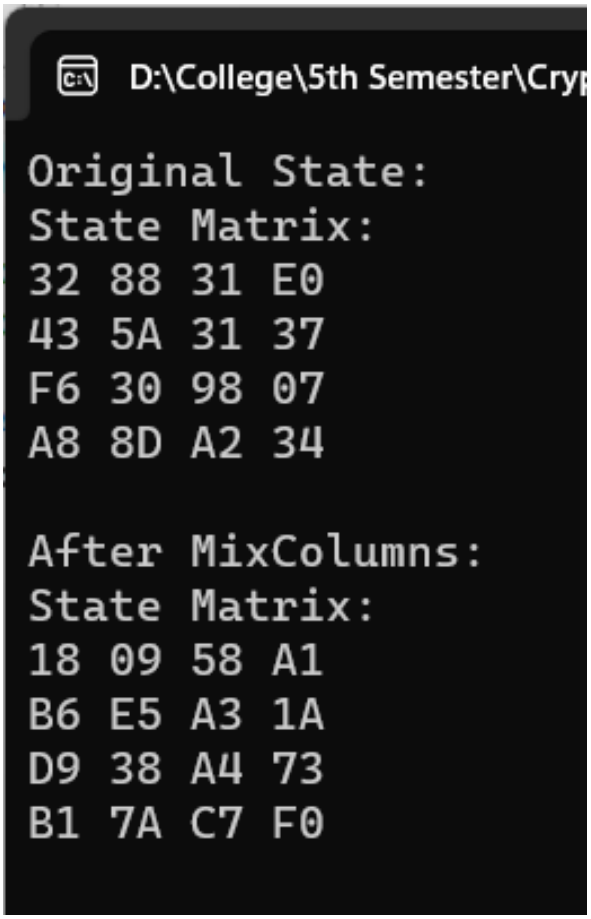
AES MixColumns step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

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("Original State:\n");
    displayState(state);

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

    return 0;
}
```

Output:



Conclusion:

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