Practical 09

Problem statement : Implement encryption and decryption using Simplified-DES scheme.

#include <stdio.h>

#include <stdint.h>

// Permutation tables

int8\_t IP[] = {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};

int8\_t IP\_inv[] = {40, 8, 48, 16, 56, 24, 64, 32,

                   39, 7, 47, 15, 55, 23, 63, 31,

                   38, 6, 46, 14, 54, 22, 62, 30,

                   37, 5, 45, 13, 53, 21, 61, 29,

                   36, 4, 44, 12, 52, 20, 60, 28,

                   35, 3, 43, 11, 51, 19, 59, 27,

                   34, 2, 42, 10, 50, 18, 58, 26,

                   33, 1, 41, 9, 49, 17, 57, 25};

// Expansion permutation table

int8\_t EP[] = {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};

// S-Boxes for substitution

int8\_t S[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, 2, 8, 4, 7, 6, 12},

        {13, 7, 0, 9, 5, 12, 3, 2, 14, 1, 4, 11, 10, 15, 6, 8},

        {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 5, 11, 14, 2, 12, 3},

        {7, 11, 4, 1, 10, 6, 0, 5, 2, 13, 14, 3, 12, 9, 8, 15}

    },

    {

        {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},

        {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9},

        {10, 3, 2, 5, 7, 15, 0, 14, 1, 13, 9, 8, 12, 6, 4, 11},

        {2, 12, 4, 1, 7, 10, 13, 6, 0, 11, 9, 5, 14, 3, 15, 8}

    },

    {

        {2, 12, 4, 1, 7, 10, 13, 6, 0, 11, 9, 5, 14, 3, 15, 8},

        {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 5, 11, 14, 2, 12, 3},

        {7, 11, 4, 1, 10, 6, 0, 5, 2, 13, 14, 3, 12, 9, 8, 15},

        {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}

    },

    {

        {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},

        {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9},

        {10, 3, 2, 5, 7, 15, 0, 14, 1, 13, 9, 8, 12, 6, 4, 11},

        {2, 12, 4, 1, 7, 10, 13, 6, 0, 11, 9, 5, 14, 3, 15, 8}

    },

    {

        {2, 12, 4, 1, 7, 10, 13, 6, 0, 11, 9, 5, 14, 3, 15, 8},

        {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 5, 11, 14, 2, 12, 3},

        {7, 11, 4, 1, 10, 6, 0, 5, 2, 13, 14, 3, 12, 9, 8, 15},

        {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9}

    },

};

// Permute input based on a table

uint64\_t permute\_64(uint64\_t input, int8\_t\* table, int size) {

    uint64\_t output = 0;

    for (int i = 0; i < size; i++) {

        output <<= 1;

        output |= (input >> (64 - table[i])) & 1;

    }

    return output;

}

// Left circular shift for key scheduling

uint64\_t left\_shift\_28(uint64\_t input, int shift) {

    return ((input << shift) | (input >> (28 - shift))) & 0xFFFFFFF;

}

// Key schedule for generating round keys

void key\_schedule(uint64\_t key, uint64\_t\* K1, uint64\_t\* K2) {

    uint64\_t left = (key >> 28) & 0xFFFFFFF;  // Left half of the key

    uint64\_t right = key & 0xFFFFFFF;         // Right half of the key

    // Generate K1

    left = left\_shift\_28(left, 1);

    right = left\_shift\_28(right, 1);

    \*K1 = (left << 28) | right;

    // Generate K2

    left = left\_shift\_28(left, 2);

    right = left\_shift\_28(right, 2);

    \*K2 = (left << 28) | right;

}

// S-Box substitution

uint8\_t s\_box\_6bit(uint8\_t input, int sbox\_index) {

    int row = ((input & 0x20) >> 4) | (input & 0x1);  // Row from bits 1 and 6

    int col = (input >> 1) & 0xF;                      // Column from bits 2 to 5

    return S[sbox\_index][row][col];                   // Get output from S-Box

}

// F-function

uint32\_t f\_function(uint32\_t right, uint64\_t subkey) {

    // Expansion permutation

    uint64\_t expanded\_right = permute\_64(right, EP, 48);

    // XOR with subkey

    expanded\_right ^= subkey;

    // S-Box substitution

    uint32\_t output = 0;

    for (int i = 0; i < 8; i++) {

        uint8\_t s\_input = (expanded\_right >> (42 - i \* 6)) & 0x3F;  // Take 6 bits

        output <<= 4;  // Prepare for 4-bit output

        output |= s\_box\_6bit(s\_input, i);  // Get output from S-Box

    }

    // P4 permutation

    return output;  // For simplicity, returning output directly

}

// DES round function

uint64\_t des\_round(uint64\_t input, uint64\_t subkey) {

    uint32\_t left = (input >> 32) & 0xFFFFFFFF;  // Left half

    uint32\_t right = input & 0xFFFFFFFF;          // Right half

    // Apply the f-function and XOR with left half

    uint32\_t new\_left = right;

    uint32\_t new\_right = left ^ f\_function(right, subkey);

    return ((uint64\_t)new\_left << 32) | new\_right;  // Return combined result

}

// DES encryption/decryption

uint64\_t des(uint64\_t input, uint64\_t K1, uint64\_t K2, int encrypt) {

    // Initial permutation

    uint64\_t permuted = permute\_64(input, IP, 64);

    // First round

    permuted = des\_round(permuted, encrypt ? K1 : K2);

    // Swap halves

    uint64\_t swapped = (permuted << 32) | (permuted >> 32);

    // Second round

    permuted = des\_round(swapped, encrypt ? K2 : K1);

    // Inverse initial permutation

    return permute\_64(permuted, IP\_inv, 64);

}

int main() {

    // 64-bit key and plaintext

    uint64\_t key = 0xAABB09182736CCDD;  // Example 64-bit key

    uint64\_t plaintext = 0x0123456789ABCDEF;  // Example 64-bit plaintext

    // Generate round keys

    uint64\_t K1, K2;

    key\_schedule(key, &K1, &K2);

    // Encrypt plaintext

    uint64\_t ciphertext = des(plaintext, K1, K2, 1);

    printf("Encrypted ciphertext: %016lX\n", ciphertext);

    uint64\_t decrypted = des(ciphertext, K1, K2, 0);

    printf("Decrypted plaintext: %016lX\n", decrypted);

    return 0;

}