#include <stdio.h>

#include <stdint.h>

#include <string.h>

// Initial Permutation Table

int 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

};

// Final Permutation Table (Inverse of IP)

int FP[] = {

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

};

// Simple function to apply permutation

uint64\_t permute(uint64\_t input, const int \*table, int tableSize) {

uint64\_t output = 0;

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

output <<= 1;

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

}

return output;

}

// Feistel function (simplified version for demo)

uint32\_t feistel(uint32\_t halfBlock, uint64\_t subkey) {

return (halfBlock ^ (uint32\_t)(subkey & 0xFFFFFFFF));

}

// Key schedule placeholder: generate same key for all rounds (for simplicity)

void generateSubkeys(uint64\_t key, uint64\_t subkeys[16]) {

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

subkeys[i] = key;

}

}

// DES encryption/decryption function

uint64\_t des(uint64\_t input, uint64\_t key, int mode) {

uint64\_t subkeys[16];

generateSubkeys(key, subkeys);

if (mode == 0) { // decrypt

for (int i = 0; i < 8; i++) {

uint64\_t temp = subkeys[i];

subkeys[i] = subkeys[15 - i];

subkeys[15 - i] = temp;

}

}

// Initial permutation

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

uint32\_t L = (uint32\_t)(permutedInput >> 32);

uint32\_t R = (uint32\_t)(permutedInput & 0xFFFFFFFF);

// 16 rounds

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

uint32\_t temp = R;

R = L ^ feistel(R, subkeys[i]);

L = temp;

}

// Combine and swap halves

uint64\_t combined = ((uint64\_t)R << 32) | L;

// Final permutation

uint64\_t finalOutput = permute(combined, FP, 64);

return finalOutput;

}

// Helper function to print 64-bit data in binary

void printBinary(uint64\_t data) {

for (int i = 63; i >= 0; i--) {

printf("%d", (data >> i) & 1);

if (i % 8 == 0) printf(" ");

}

printf("\n");

}

// Convert string to 64-bit block (pad with 0s)

uint64\_t strToBlock(char \*str) {

uint64\_t block = 0;

for (int i = 0; i < 8 && str[i]; i++) {

block |= ((uint64\_t)(unsigned char)str[i]) << (56 - 8 \* i);

}

return block;

}

// Convert 64-bit block back to string

void blockToStr(uint64\_t block, char \*out) {

for (int i = 0; i < 8; i++) {

out[i] = (char)((block >> (56 - 8 \* i)) & 0xFF);

}

out[8] = '\0';

}

int main() {

char inputText[9];

char keyText[9];

printf("Enter 8-character plaintext: ");

scanf("%8s", inputText);

printf("Enter 8-character key: ");

scanf("%8s", keyText);

uint64\_t plaintext = strToBlock(inputText);

uint64\_t key = strToBlock(keyText);

printf("Plaintext binary: ");

printBinary(plaintext);

uint64\_t encrypted = des(plaintext, key, 1);

printf("Encrypted binary: ");

printBinary(encrypted);

char encryptedStr[9];

blockToStr(encrypted, encryptedStr);

printf("Encrypted text: %s\n", encryptedStr);

uint64\_t decrypted = des(encrypted, key, 0);

printf("Decrypted binary: ");

printBinary(decrypted);

char decryptedStr[9];

blockToStr(decrypted, decryptedStr);

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

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.