22. Write a C program for Encrypt and decrypt in cipher block chaining mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES, DES. Test data for S-DES using a binary initialization vector of 1010 1010. A binary plaintext of 0000 0001 0010 0011 encrypted with a binary key of 01111 11101 should give a binary plaintext of 1111 0100 0000 1011. Decryption should work correspondingly.

#include <stdio.h>

#include <stdint.h>

uint8\_t IP[] = {1, 5, 2, 0, 3, 7, 4, 6};

uint8\_t IP\_INV[] = {3, 0, 2, 4, 6, 1, 7, 5};

uint8\_t EP[] = {3, 0, 1, 2, 1, 2, 3, 0};

uint8\_t P4[] = {1, 3, 2, 0};

uint8\_t P10[] = {2, 4, 1, 6, 3, 9, 0, 8, 7, 5};

uint8\_t P8[] = {5, 2, 6, 3, 7, 4, 9, 8};

uint8\_t S0[4][4] = {

{1, 0, 3, 2},

{3, 2, 1, 0},

{0, 2, 1, 3},

{3, 1, 3, 2}

};

uint8\_t S1[4][4] = {

{0, 1, 2, 3},

{2, 0, 1, 3},

{3, 0, 1, 0},

{2, 1, 0, 3}

};

uint8\_t permute(uint8\_t in, uint8\_t\* p, int n) {

uint8\_t out = 0;

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

out <<= 1;

out |= (in >> (7 - p[i])) & 1;

}

return out;

}

uint8\_t leftShift5(uint8\_t in, int shifts) {

return ((in << shifts) | (in >> (5 - shifts))) & 0x1F;

}

uint16\_t keyGen(uint16\_t key, uint8\_t\* k1, uint8\_t\* k2) {

uint16\_t perm = 0;

for (int i = 0; i < 10; i++) {

perm <<= 1;

perm |= (key >> (9 - P10[i])) & 1;

}

uint8\_t left = (perm >> 5) & 0x1F;

uint8\_t right = perm & 0x1F;

left = leftShift5(left, 1);

right = leftShift5(right, 1);

uint16\_t merged = (left << 5) | right;

\*k1 = 0;

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

\*k1 <<= 1;

\*k1 |= (merged >> (9 - P8[i])) & 1;

}

left = leftShift5(left, 2);

right = leftShift5(right, 2);

merged = (left << 5) | right;

\*k2 = 0;

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

\*k2 <<= 1;

\*k2 |= (merged >> (9 - P8[i])) & 1;

}

return 0;

}

uint8\_t sbox(uint8\_t in, uint8\_t box[4][4]) {

uint8\_t row = ((in & 0x8) >> 2) | (in & 0x1);

uint8\_t col = (in >> 1) & 0x3;

return box[row][col];

}

uint8\_t f(uint8\_t r, uint8\_t sk) {

uint8\_t ep = 0;

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

ep <<= 1;

ep |= (r >> (3 - EP[i])) & 1;

}

uint8\_t x = ep ^ sk;

uint8\_t left = (x >> 4) & 0xF;

uint8\_t right = x & 0xF;

uint8\_t out = (sbox(left, S0) << 2) | sbox(right, S1);

uint8\_t p4out = 0;

for (int i = 0; i < 4; i++) {

p4out <<= 1;

p4out |= (out >> (3 - P4[i])) & 1;

}

return p4out;

}

uint8\_t fk(uint8\_t in, uint8\_t k1, uint8\_t k2, int isDecrypt) {

uint8\_t ip = permute(in, IP, 8);

uint8\_t left = ip >> 4;

uint8\_t right = ip & 0xF;

uint8\_t t1 = f(right, isDecrypt ? k2 : k1);

left ^= t1;

uint8\_t swapped = (right << 4) | left;

right = swapped & 0xF;

left = swapped >> 4;

uint8\_t t2 = f(right, isDecrypt ? k1 : k2);

left ^= t2;

uint8\_t preout = (left << 4) | right;

uint8\_t out = permute(preout, IP\_INV, 8);

return out;

}

void encryptCBC(uint8\_t\* pt, uint8\_t\* ct, int n, uint8\_t k1, uint8\_t k2, uint8\_t iv) {

uint8\_t prev = iv;

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

uint8\_t block = pt[i] ^ prev;

ct[i] = fk(block, k1, k2, 0);

prev = ct[i];

}

}

void decryptCBC(uint8\_t\* ct, uint8\_t\* pt, int n, uint8\_t k1, uint8\_t k2, uint8\_t iv) {

uint8\_t prev = iv;

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

uint8\_t temp = fk(ct[i], k1, k2, 1);

pt[i] = temp ^ prev;

prev = ct[i];

}

}

int main() {

uint8\_t plaintext[2] = {0x01, 0x23};

uint8\_t ciphertext[2], decrypted[2];

uint8\_t iv = 0xAA;

uint8\_t k1, k2;

uint16\_t key = 0x1FD;

keyGen(key, &k1, &k2);

encryptCBC(plaintext, ciphertext, 2, k1, k2, iv);

printf("Encrypted: %02X %02X\n", ciphertext[0], ciphertext[1]);

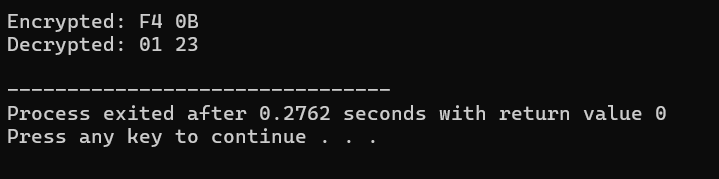
decryptCBC(ciphertext, decrypted, 2, k1, k2, iv);

printf("Decrypted: %02X %02X\n", decrypted[0], decrypted[1]);

return 0;

}

Output

s