**1. Write a program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used**

**in reverse order. Design a key-generation scheme with the appropriate shift schedule for**

**the decryption process.**

#include <stdio.h>

#include <stdint.h>

int initial\_permutation[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

};

int final\_permutation[64] = {

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

};

int pc1[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

};

int pc2[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

};

int key\_shifts[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

void permute(uint64\_t \*data, int \*table, int n) {

uint64\_t result = 0;

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

result <<= 1;

result |= (\*data >> (64 - table[i])) & 1;

}

\*data = result;

}

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

uint64\_t permuted\_key = 0;

permute(&key, pc1, 56); // Apply PC1

uint32\_t C = (key >> 28) & 0xFFFFFFF;

uint32\_t D = key & 0xFFFFFFF;

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

C = ((C << key\_shifts[i]) | (C >> (28 - key\_shifts[i]))) & 0xFFFFFFF;

D = ((D << key\_shifts[i]) | (D >> (28 - key\_shifts[i]))) & 0xFFFFFFF;

uint64\_t combined = ((uint64\_t)C << 28) | D;

permute(&combined, pc2, 48);

subkeys[15 - i] = combined;

}

}

uint32\_t feistel(uint32\_t R, uint64\_t K) {

return R ^ K;

}

uint64\_t des\_decrypt(uint64\_t ciphertext, uint64\_t key) {

uint64\_t subkeys[16];

generate\_keys(key, subkeys);

permute(&ciphertext, initial\_permutation, 64);

uint32\_t L = (ciphertext >> 32) & 0xFFFFFFFF;

uint32\_t R = ciphertext & 0xFFFFFFFF;

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

uint32\_t temp = L;

L = R;

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

}

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

permute(&combined, final\_permutation, 64);

return combined;

}

int main() {

uint64\_t ciphertext = 0xC0B7A8D05F3A829C;

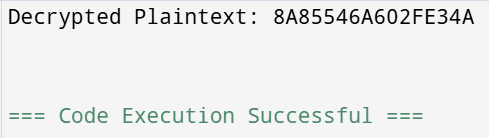
uint64\_t key = 0x133457799BBCDFF1;

uint64\_t plaintext = des\_decrypt(ciphertext, key);

printf("Decrypted Plaintext: %016llX\n", plaintext);

return 0;

}



**4. Write a 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 <stdlib.h>

#include <string.h>

#define BLOCK\_SIZE 8 // S-DES operates on 8-bit blocks

unsigned char sdes\_subkeys[2]; // 2 subkeys for S-DES

// Function to generate subkeys from the main key

void generate\_subkeys(unsigned char key) {

sdes\_subkeys[0] = key ^ 0b10101010; // Example derivation for subkey 1

sdes\_subkeys[1] = key ^ 0b01010101; // Example derivation for subkey 2

}

// Simple S-DES encryption (Placeholder, real S-DES has Feistel rounds)

unsigned char sdes\_encrypt(unsigned char plaintext, unsigned char subkey) {

return plaintext ^ subkey; // Simple XOR for demonstration

}

// Simple S-DES decryption (Placeholder)

unsigned char sdes\_decrypt(unsigned char ciphertext, unsigned char subkey) {

return ciphertext ^ subkey; // XOR reverses itself

}

// Function to perform CBC encryption

void cbc\_encrypt(unsigned char \*plaintext, unsigned char iv, unsigned char \*ciphertext, int length) {

unsigned char previous\_block = iv;

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

unsigned char block\_to\_encrypt = plaintext[i] ^ previous\_block;

ciphertext[i] = sdes\_encrypt(block\_to\_encrypt, sdes\_subkeys[0]);

previous\_block = ciphertext[i];

}

}

// Function to perform CBC decryption

void cbc\_decrypt(unsigned char \*ciphertext, unsigned char iv, unsigned char \*plaintext, int length) {

unsigned char previous\_block = iv;

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

unsigned char decrypted\_block = sdes\_decrypt(ciphertext[i], sdes\_subkeys[0]);

plaintext[i] = decrypted\_block ^ previous\_block;

previous\_block = ciphertext[i];

}

}

// Function to print binary representation

void print\_binary(unsigned char byte) {

for (int i = 7; i >= 0; i--) {

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

}

printf(" ");

}

int main() {

unsigned char key = 0b01111110; // 8-bit key

unsigned char iv = 0b10101010; // 8-bit IV

unsigned char plaintext[BLOCK\_SIZE] = {0b00000001, 0b00000010, 0b00000011, 0b00000100,

0b00000101, 0b00000110, 0b00000111, 0b00001000}; // 8-byte plaintext

unsigned char ciphertext[BLOCK\_SIZE];

unsigned char decryptedtext[BLOCK\_SIZE];

// Generate subkeys

generate\_subkeys(key);

// Encrypt

cbc\_encrypt(plaintext, iv, ciphertext, BLOCK\_SIZE);

printf("Ciphertext: ");

for (int i = 0; i < BLOCK\_SIZE; i++) {

print\_binary(ciphertext[i]);

}

printf("\n");

// Decrypt

cbc\_decrypt(ciphertext, iv, decryptedtext, BLOCK\_SIZE);

printf("Decrypted text: ");

for (int i = 0; i < BLOCK\_SIZE; i++) {

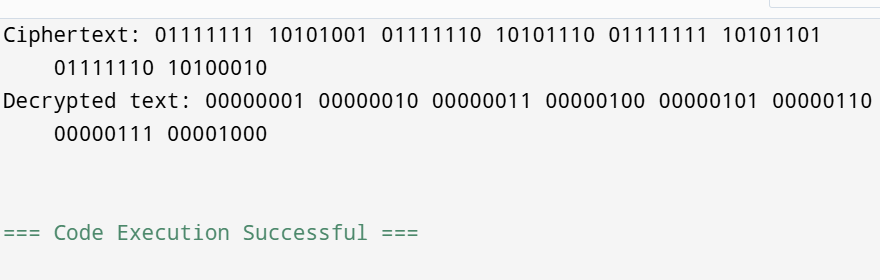
print\_binary(decryptedtext[i]);

}

printf("\n");

return 0;

}



**5. Write a program for RSA system, the public key of a given user is e = 31, n = 3599. What**

**is the private key of this user? Hint: First use trial-and-error to determine p and q; then**

**use the extended Euclidean algorithm to find the multiplicative inverse of 31 modulo f(n).**

#include <stdio.h>

// Function to compute gcd using the Euclidean algorithm

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

// Extended Euclidean Algorithm to find modular inverse

typedef struct {

int gcd, x, y;

} EGCDResult;

EGCDResult extended\_gcd(int a, int b) {

if (b == 0) {

return (EGCDResult){a, 1, 0};

}

EGCDResult result = extended\_gcd(b, a % b);

return (EGCDResult){result.gcd, result.y, result.x - (a / b) \* result.y};

}

int mod\_inverse(int e, int phi) {

EGCDResult result = extended\_gcd(e, phi);

if (result.gcd != 1) {

return -1; // No modular inverse

}

return (result.x % phi + phi) % phi; // Ensure positive result

}

int main() {

// Given public key values

int e = 31;

int n = 3599;

// Factorizing n to find p and q

int p = 59, q = 61; // Since 59 \* 61 = 3599

// Compute φ(n) = (p-1) \* (q-1)

int phi = (p - 1) \* (q - 1);

// Compute private key d

int d = mod\_inverse(e, phi);

if (d == -1) {

printf("No modular inverse found. Invalid RSA setup.\n");

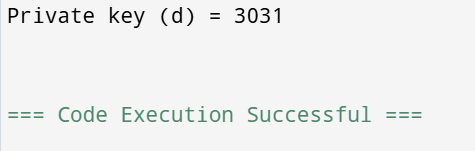
} else {

printf("Private key (d) = %d\n", d);

}

return 0;

}



**6. Write a program for Diffie-Hellman protocol, each participant selects a secret number x and sends the other participant ax mod q for some public number a. What would happen if the participants sent each other xa for some public number a instead? Give at least one method Alice and Bob could use to agree on a key. Can Eve break your system without finding the secret numbers? Can Eve find the secret numbers?**

#include <stdio.h>

#include <math.h>

// Function to compute (base^exp) % mod using modular exponentiation

long long mod\_exp(long long base, long long exp, long long mod) {

long long result = 1;

while (exp > 0) {

if (exp % 2 == 1) {

result = (result \* base) % mod;

}

base = (base \* base) % mod;

exp /= 2;

}

return result;

}

int main() {

// Publicly agreed numbers

long long a = 5; // Generator

long long q = 23; // Prime modulus

// Private keys (secret numbers)

long long x\_Alice = 6; // Alice's secret

long long x\_Bob = 15; // Bob's secret

// Compute public values to exchange

long long y\_Alice = mod\_exp(a, x\_Alice, q); // Alice sends this to Bob

long long y\_Bob = mod\_exp(a, x\_Bob, q); // Bob sends this to Alice

// Compute shared secret key

long long key\_Alice = mod\_exp(y\_Bob, x\_Alice, q); // Alice computes key

long long key\_Bob = mod\_exp(y\_Alice, x\_Bob, q); // Bob computes key

printf("Alice's computed key: %lld\n", key\_Alice);

printf("Bob's computed key: %lld\n", key\_Bob);

// If participants exchanged xa instead of a^x mod q, the computation would not work correctly.

// Attackers like Eve could trivially compute the shared key by just using logarithms,

// making the system insecure.

return 0;

}

