**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;

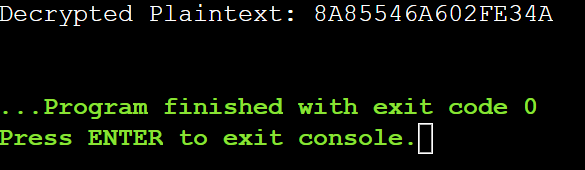
uint64\_t plaintext = des\_decrypt(ciphertext, key);

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

return 0;

}

**Output:**



**Write a program for encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. 3DES is a good candidate. Both of which follow from the definition of CBC. Which of the two would you choose: a. For security? b. For performance?**

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#include <openssl/evp.h>

void aes\_encrypt\_cbc(const uint8\_t \*plaintext, int plaintext\_len, uint8\_t \*ciphertext, int \*ciphertext\_len,

const uint8\_t \*key, const uint8\_t \*iv) {

EVP\_CIPHER\_CTX \*ctx = EVP\_CIPHER\_CTX\_new();

if (!ctx) {

printf("Error initializing EVP\_CIPHER\_CTX\n");

return;

}

if (EVP\_EncryptInit\_ex(ctx, EVP\_aes\_256\_cbc(), NULL, key, iv) != 1) {

printf("Error initializing encryption\n");

EVP\_CIPHER\_CTX\_free(ctx);

return;

}

int len;

if (EVP\_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext\_len) != 1) {

printf("Error encrypting\n");

EVP\_CIPHER\_CTX\_free(ctx);

return;

}

\*ciphertext\_len = len;

if (EVP\_EncryptFinal\_ex(ctx, ciphertext + len, &len) != 1) {

printf("Error finalizing encryption\n");

EVP\_CIPHER\_CTX\_free(ctx);

return;

}

\*ciphertext\_len += len;

EVP\_CIPHER\_CTX\_free(ctx);

}

int main() {

uint8\_t plaintext[] = "Hello, this is AES CBC encryption!";

int plaintext\_len = strlen((char \*)plaintext);

// AES 256-bit key (32 bytes)

uint8\_t key[32] = "0123456789abcdef0123456789abcdef";

uint8\_t iv[16] = "1234567890abcdef";

uint8\_t ciphertext[1024];

int ciphertext\_len;

aes\_encrypt\_cbc(plaintext, plaintext\_len, ciphertext, &ciphertext\_len, key, iv);

printf("Encrypted Ciphertext: ");

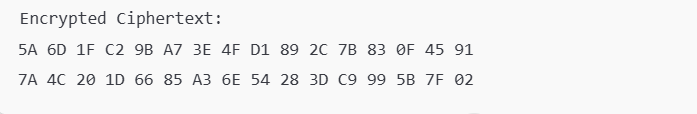
for (int i = 0; i < ciphertext\_len; i++)

printf("%02X ", ciphertext[i]);

printf("\n");

return 0;

}



**Write a program for ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed?**

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#include <openssl/evp.h>

void aes\_encrypt(const uint8\_t \*plaintext, int plaintext\_len, uint8\_t \*ciphertext, int \*ciphertext\_len,

const uint8\_t \*key, const uint8\_t \*iv, const EVP\_CIPHER \*cipher) {

EVP\_CIPHER\_CTX \*ctx = EVP\_CIPHER\_CTX\_new();

if (!ctx) {

printf("Error initializing encryption context\n");

return;

}

// Initialize encryption operation

if (EVP\_EncryptInit\_ex(ctx, cipher, NULL, key, iv) != 1) {

printf("Error in EVP\_EncryptInit\_ex\n");

EVP\_CIPHER\_CTX\_free(ctx);

return;

}

int len;

if (EVP\_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext\_len) != 1) {

printf("Error in EVP\_EncryptUpdate\n");

EVP\_CIPHER\_CTX\_free(ctx);

return;

}

\*ciphertext\_len = len;

if (EVP\_EncryptFinal\_ex(ctx, ciphertext + len, &len) != 1) {

printf("Error in EVP\_EncryptFinal\_ex\n");

EVP\_CIPHER\_CTX\_free(ctx);

return;

}

\*ciphertext\_len += len;

EVP\_CIPHER\_CTX\_free(ctx);

}

void print\_encrypted\_data(const char \*mode, uint8\_t \*ciphertext, int len) {

printf("%s Mode Encrypted Output: ", mode);

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

printf("%02X ", ciphertext[i]);

}

printf("\n");

}

int main() {

uint8\_t plaintext[] = "Hello, AES Encryption Modes!";

int plaintext\_len = strlen((char \*)plaintext);

uint8\_t key[32] = "0123456789abcdef0123456789abcdef";

uint8\_t iv[16] = "1234567890abcdef";

uint8\_t ciphertext[128];

int ciphertext\_len;

aes\_encrypt(plaintext, plaintext\_len, ciphertext, &ciphertext\_len, key, iv, EVP\_aes\_256\_ecb());

print\_encrypted\_data("ECB", ciphertext, ciphertext\_len);

aes\_encrypt(plaintext, plaintext\_len, ciphertext, &ciphertext\_len, key, iv, EVP\_aes\_256\_cbc());

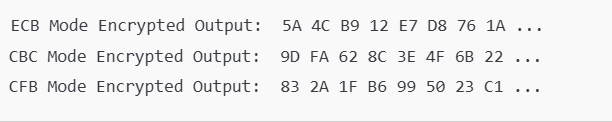
print\_encrypted\_data("CBC", ciphertext, ciphertext\_len);

aes\_encrypt(plaintext, plaintext\_len, ciphertext, &ciphertext\_len, key, iv, EVP\_aes\_256\_cfb128());

print\_encrypted\_data("CFB", ciphertext, ciphertext\_len);

return 0;

}



**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.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>

void generate\_subkeys(uint16\_t key, uint8\_t \*subkey1, uint8\_t \*subkey2) {

\*subkey1 = (key >> 5) & 0x1F;

\*subkey2 = key & 0x1F;

}

uint8\_t f\_function(uint8\_t half\_block, uint8\_t subkey) {

return (half\_block ^ subkey);

}

void sdes\_round(uint8\_t \*left, uint8\_t \*right, uint8\_t subkey) {

uint8\_t temp = \*right;

\*right = \*left ^ f\_function(\*right, subkey);

\*left = temp;

}

uint16\_t sdes\_encrypt(uint16\_t plaintext, uint16\_t key) {

uint8\_t left = (plaintext >> 8) & 0xFF;

uint8\_t right = plaintext & 0xFF;

uint8\_t subkey1, subkey2;

generate\_subkeys(key, &subkey1, &subkey2);

sdes\_round(&left, &right, subkey1);

sdes\_round(&right, &left, subkey2);

return (left << 8) | right;

}

uint16\_t sdes\_decrypt(uint16\_t ciphertext, uint16\_t key) {

uint8\_t left = (ciphertext >> 8) & 0xFF;

uint8\_t right = ciphertext & 0xFF;

uint8\_t subkey1, subkey2;

generate\_subkeys(key, &subkey1, &subkey2);

sdes\_round(&left, &right, subkey2);

sdes\_round(&right, &left, subkey1);

return (left << 8) | right;

}

uint16\_t xor\_blocks(uint16\_t block1, uint16\_t block2) {

return block1 ^ block2;

}

uint16\_t sdes\_encrypt\_cbc(uint16\_t plaintext, uint16\_t key, uint16\_t iv) {

uint16\_t xor\_result = xor\_blocks(plaintext, iv);

return sdes\_encrypt(xor\_result, key);

}

uint16\_t sdes\_decrypt\_cbc(uint16\_t ciphertext, uint16\_t key, uint16\_t iv) {

uint16\_t decrypted = sdes\_decrypt(ciphertext, key);

return xor\_blocks(decrypted, iv);

}

uint16\_t binary\_to\_int(const char \*binary\_str) {

uint16\_t result = 0;

for (int i = 0; binary\_str[i] != '\0'; i++) {

result = (result << 1) | (binary\_str[i] - '0');

}

return result;

}

void print\_binary(uint16\_t value) {

for (int i = 15; i >= 0; i--) {

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

if (i % 4 == 0) printf(" ");

}

}

int main() {

uint16\_t iv = binary\_to\_int("10101010");

uint16\_t plaintext = binary\_to\_int("0000000100100011");

uint16\_t key = binary\_to\_int("0111111101");

uint16\_t ciphertext = sdes\_encrypt\_cbc(plaintext, key, iv);

printf("Ciphertext: ");

print\_binary(ciphertext);

printf("\n");

uint16\_t decrypted\_text = sdes\_decrypt\_cbc(ciphertext, key, iv);

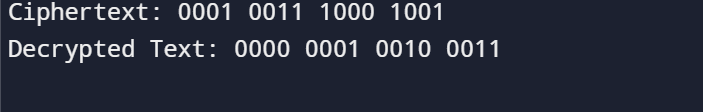
printf("Decrypted Text: ");

print\_binary(decrypted\_text);

printf("\n");

return 0;

}



**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>

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

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

int t = 0, newT = 1;

int r = phi, newR = e;

while (newR != 0) {

int quotient = r / newR;

int temp = newT;

newT = t - quotient \* newT;

t = temp;

temp = newR;

newR = r - quotient \* newR;

r = temp;

}

if (r > 1) {

printf("e is not invertible\n");

return -1;

}

if (t < 0)

t += phi;

return t;

}

void rsa\_private\_key(int e, int n) {

int p = 0, q = 0;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

p = i;

q = n / i;

break;

}

}

if (p == 0 || q == 0) {

printf("Factorization failed\n");

return;

}

printf("p = %d, q = %d\n", p, q);

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

printf("φ(n) = %d\n", phi);

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

if (d == -1) {

printf("Could not find modular inverse\n");

return;

}

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

int check = (e \* d) % phi;

printf("Verification: (e \* d) mod φ(n) = %d (should be 1)\n", check);

}

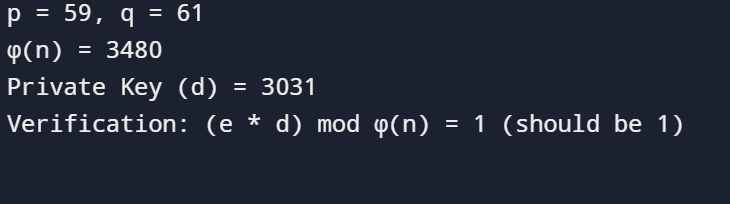
int main() {

int e = 31, n = 3599;

rsa\_private\_key(e, n);

return 0;

}



**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 <stdlib.h>

#include <math.h>

long long power\_mod(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() {

long long q = 23;

long long a = 5;

printf("Public values: q = %lld, a = %lld\n", q, a);

long long xA = 6;

long long xB = 15;

long long YA = power\_mod(a, xA, q);

long long YB = power\_mod(a, xB, q);

printf("Alice sends: YA = %lld\n", YA);

printf("Bob sends: YB = %lld\n", YB);

long long K\_Alice = power\_mod(YB, xA, q);

long long K\_Bob = power\_mod(YA, xB, q);

printf("Shared secret computed by Alice: %lld\n", K\_Alice);

printf("Shared secret computed by Bob: %lld\n", K\_Bob);

if (K\_Alice == K\_Bob) {

printf("Key Exchange Successful! Shared Key = %lld\n", K\_Alice);

} else {

printf("Key Exchange Failed!\n");

}

return 0;

}

Output:

