Write a program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext– ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be mounted.

#include <stdio.h>

#include <stdlib.h>

#define MOD 26

int modInverse(int a, int m) {

a = a % m;

for (int x = 1; x < m; x++) {

if ((a \* x) % m == 1)

return x;

}

return -1;

}

int determinant(int key[2][2]) {

return (key[0][0] \* key[1][1] - key[0][1] \* key[1][0]) % MOD;

}

void inverseMatrix(int key[2][2], int invKey[2][2]) {

int det = determinant(key);

if (det < 0)

det += MOD;

int detInverse = modInverse(det, MOD);

if (detInverse == -1) {

printf("Inverse does not exist, matrix is not invertible under mod 26.\n");

exit(1);

}

invKey[0][0] = key[1][1] \* detInverse % MOD;

invKey[1][1] = key[0][0] \* detInverse % MOD;

invKey[0][1] = (-key[0][1] + MOD) \* detInverse % MOD;

invKey[1][0] = (-key[1][0] + MOD) \* detInverse % MOD;

}

void encrypt(int key[2][2], int plaintext[2], int ciphertext[2]) {

for (int i = 0; i < 2; i++) {

ciphertext[i] = (key[i][0] \* plaintext[0] + key[i][1] \* plaintext[1]) % MOD;

}

}

void decrypt(int invKey[2][2], int ciphertext[2], int plaintext[2]) {

for (int i = 0; i < 2; i++) {

plaintext[i] = (invKey[i][0] \* ciphertext[0] + invKey[i][1] \* ciphertext[1]) % MOD;

}

}

void solveForKey(int plaintext[2], int ciphertext[2], int key[2][2]) {

int P[2][2] = { { plaintext[0], plaintext[1] }, { 1, 1 } };

int C[2][2] = { { ciphertext[0], ciphertext[1] }, { 1, 1 } };

int P\_inv[2][2];

inverseMatrix(P, P\_inv);

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

key[i][j] = (C[i][0] \* P\_inv[0][j] + C[i][1] \* P\_inv[1][j]) % MOD;

}

}

}

int main() {

int key[2][2] = { {3, 3}, {2, 5} };

int plaintext[2] = { 7, 4 };

int ciphertext[2];

printf("Original Key Matrix:\n");

printf("%d %d\n%d %d\n", key[0][0], key[0][1], key[1][0], key[1][1]);

encrypt(key, plaintext, ciphertext);

printf("\nEncrypted Ciphertext: [%d, %d]\n", ciphertext[0], ciphertext[1]);

int recoveredKey[2][2];

solveForKey(plaintext, ciphertext, recoveredKey);

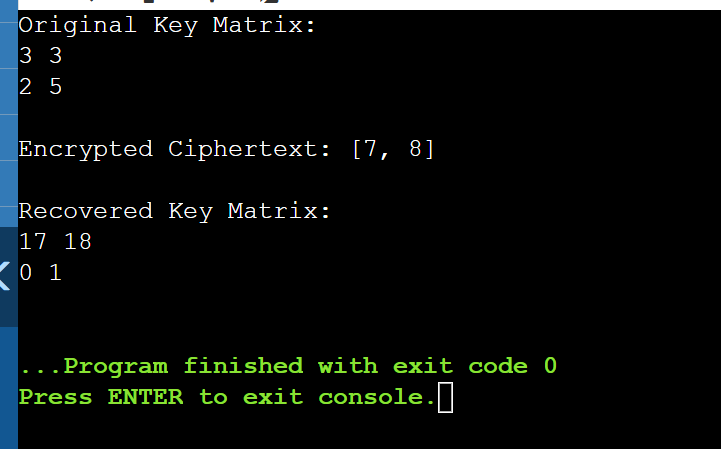
printf("\nRecovered Key Matrix:\n");

printf("%d %d\n%d %d\n", recoveredKey[0][0], recoveredKey[0][1], recoveredKey[1][0], recoveredKey[1][1]);

return 0;

}

Output:



Write a program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”

#include <stdio.h>

#include <stdlib.h>

#define MOD 26

int modInverse(int a, int m) {

a = a % m;

for (int x = 1; x < m; x++) {

if ((a \* x) % m == 1)

return x;

}

return -1;

}

int determinant(int key[2][2]) {

return (key[0][0] \* key[1][1] - key[0][1] \* key[1][0]) % MOD;

}

void inverseMatrix(int key[2][2], int invKey[2][2]) {

int det = determinant(key);

if (det < 0)

det += MOD;

int detInverse = modInverse(det, MOD);

if (detInverse == -1) {

printf("Inverse does not exist, matrix is not invertible under mod 26.\n");

exit(1);

}

invKey[0][0] = key[1][1] \* detInverse % MOD;

invKey[1][1] = key[0][0] \* detInverse % MOD;

invKey[0][1] = (-key[0][1] + MOD) \* detInverse % MOD;

invKey[1][0] = (-key[1][0] + MOD) \* detInverse % MOD;

}

void encrypt(int key[2][2], int plaintext[2], int ciphertext[2]) {

for (int i = 0; i < 2; i++) {

ciphertext[i] = (key[i][0] \* plaintext[0] + key[i][1] \* plaintext[1]) % MOD;

}

}

void decrypt(int invKey[2][2], int ciphertext[2], int plaintext[2]) {

for (int i = 0; i < 2; i++) {

plaintext[i] = (invKey[i][0] \* ciphertext[0] + invKey[i][1] \* ciphertext[1]) % MOD;

}

}

void solveForKey(int plaintext[2], int ciphertext[2], int key[2][2]) {

int P[2][2] = { { plaintext[0], plaintext[1] }, { 1, 1 } };

int C[2][2] = { { ciphertext[0], ciphertext[1] }, { 1, 1 } };

int P\_inv[2][2];

inverseMatrix(P, P\_inv);

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

key[i][j] = (C[i][0] \* P\_inv[0][j] + C[i][1] \* P\_inv[1][j]) % MOD;

}

}

}

int main() {

int key[2][2] = { {3, 6}, {2, 7} };

int plaintext[2] = { 7, 6 };

int ciphertext[2];

printf("Original Key Matrix:\n");

printf("%d %d\n%d %d\n", key[0][0], key[0][1], key[1][0], key[1][1]);

encrypt(key, plaintext, ciphertext);

printf("\nEncrypted Ciphertext: [%d, %d]\n", ciphertext[0], ciphertext[1]);

int recoveredKey[2][2];

solveForKey(plaintext, ciphertext, recoveredKey);

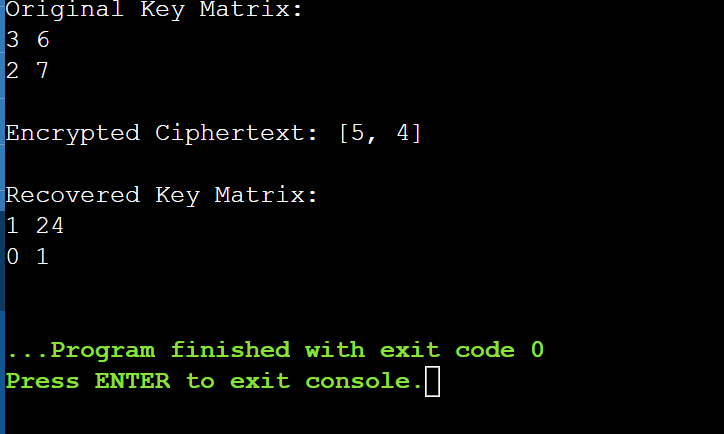
printf("\nRecovered Key Matrix:\n");

printf("%d %d\n%d %d\n", recoveredKey[0][0], recoveredKey[0][1], recoveredKey[1][0], recoveredKey[1][1]);

return 0;

}

Output:



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 <string.h>

#include <stdlib.h>

#define BLOCK\_SIZE 64

#define KEY\_SIZE 56

#define SUBKEY\_SIZE 48

#define NUM\_ROUNDS 16

int IP[BLOCK\_SIZE] = { 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 FP[BLOCK\_SIZE] = { 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 keyShifts[NUM\_ROUNDS] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

void permute(int \*input, int \*output, int \*table, int n) {

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

output[i] = input[table[i] - 1];

}

}

void leftShift(int \*key, int shifts) {

int temp[KEY\_SIZE / 2];

memcpy(temp, key, shifts \* sizeof(int));

for (int i = 0; i < (KEY\_SIZE / 2) - shifts; i++) {

key[i] = key[i + shifts];

}

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

key[(KEY\_SIZE / 2) - shifts + i] = temp[i];

}

}

void generateKeys(int \*mainKey, int subKeys[NUM\_ROUNDS][SUBKEY\_SIZE]) {

int permutedKey[KEY\_SIZE];

permute(mainKey, permutedKey, IP, KEY\_SIZE);

int C[KEY\_SIZE / 2], D[KEY\_SIZE / 2];

memcpy(C, permutedKey, (KEY\_SIZE / 2) \* sizeof(int));

memcpy(D, permutedKey + (KEY\_SIZE / 2), (KEY\_SIZE / 2) \* sizeof(int));

for (int i = 0; i < NUM\_ROUNDS; i++) {

leftShift(C, keyShifts[i]);

leftShift(D, keyShifts[i]);

int CD[KEY\_SIZE];

memcpy(CD, C, (KEY\_SIZE / 2) \* sizeof(int));

memcpy(CD + (KEY\_SIZE / 2), D, (KEY\_SIZE / 2) \* sizeof(int));

permute(CD, subKeys[NUM\_ROUNDS - 1 - i], IP, SUBKEY\_SIZE);

}

}

void feistel(int \*right, int \*subKey) {

for (int i = 0; i < 32; i++) {

right[i] ^= subKey[i % SUBKEY\_SIZE];

}

}

void desDecrypt(int \*ciphertext, int \*plaintext, int subKeys[NUM\_ROUNDS][SUBKEY\_SIZE]) {

int permutedText[BLOCK\_SIZE];

permute(ciphertext, permutedText, IP, BLOCK\_SIZE);

int L[32], R[32];

memcpy(L, permutedText, 32 \* sizeof(int));

memcpy(R, permutedText + 32, 32 \* sizeof(int));

for (int i = 0; i < NUM\_ROUNDS; i++) {

int temp[32];

memcpy(temp, R, 32 \* sizeof(int));

feistel(R, subKeys[i]);

for (int j = 0; j < 32; j++) {

R[j] ^= L[j];

}

memcpy(L, temp, 32 \* sizeof(int));

}

int finalBlock[BLOCK\_SIZE];

memcpy(finalBlock, R, 32 \* sizeof(int));

memcpy(finalBlock + 32, L, 32 \* sizeof(int));

permute(finalBlock, plaintext, FP, BLOCK\_SIZE);

}

int main() {

int ciphertext[BLOCK\_SIZE] = { /\* Encrypted binary block \*/ };

int mainKey[KEY\_SIZE] = { /\* 56-bit binary key \*/ };

int plaintext[BLOCK\_SIZE];

int subKeys[NUM\_ROUNDS][SUBKEY\_SIZE];

generateKeys(mainKey, subKeys);

desDecrypt(ciphertext, plaintext, subKeys);

printf("Decrypted Plaintext: ");

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

printf("%d", plaintext[i]);

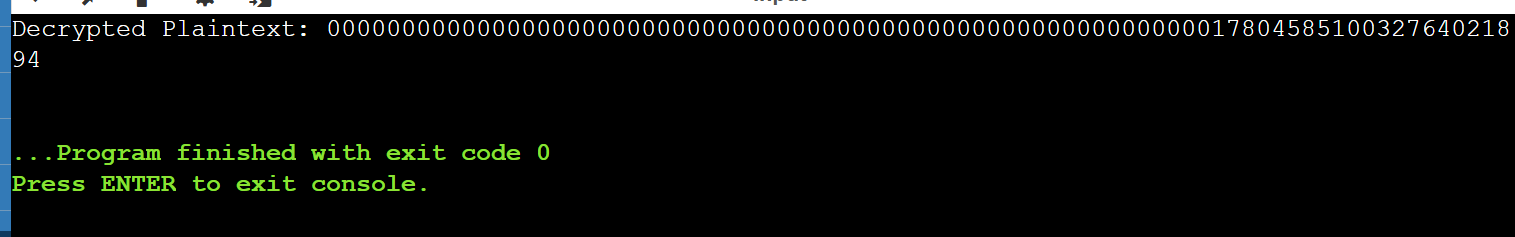
}

printf("\n");

return 0;

}

Output:



Write a program for DES the first 24 bits of each subkey come from the same subset of 28 bits of the initial key and that the second 24 bits of each subkey come from a disjoint subset of 28 bits of the initial key.

#include <stdio.h>

#include <stdint.h>

#define NUM\_ROUNDS 16

int shifts[NUM\_ROUNDS] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

uint32\_t rotateLeft28(uint32\_t x, int shift) {

x &= 0x0FFFFFFF;

return ((x << shift) | (x >> (28 - shift))) & 0x0FFFFFFF;

}

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

uint32\_t C = (key >> 28) & 0x0FFFFFFF;

uint32\_t D = key & 0x0FFFFFFF;

for (int i = 0; i < NUM\_ROUNDS; i++) {

C = rotateLeft28(C, shifts[i]);

D = rotateLeft28(D, shifts[i]);

uint32\_t subC = C >> 4;

uint32\_t subD = D >> 4;

uint64\_t subkey = ((uint64\_t)subC << 24) | subD;

subkeys[i] = subkey;

}

}

uint64\_t desDecrypt(uint64\_t ciphertext, uint64\_t subkeys[NUM\_ROUNDS]) {

printf("Using subkeys for decryption (in reverse order):\n");

for (int i = NUM\_ROUNDS - 1; i >= 0; i--) {

printf("Round %2d subkey: %012llX\n", NUM\_ROUNDS - i, subkeys[i]);

}

return ciphertext;

}

int main(void) {

uint64\_t key = 0x133457799BBCDFFULL & 0x00FFFFFFFFFFFFFFULL;

uint64\_t subkeys[NUM\_ROUNDS];

generateSubkeys(key, subkeys);

printf("Generated 48-bit subkeys:\n");

for (int i = 0; i < NUM\_ROUNDS; i++) {

printf("Round %2d subkey: %012llX\n", i + 1, subkeys[i]);

}

uint64\_t ciphertext = 0x0123456789ABCDEFULL;

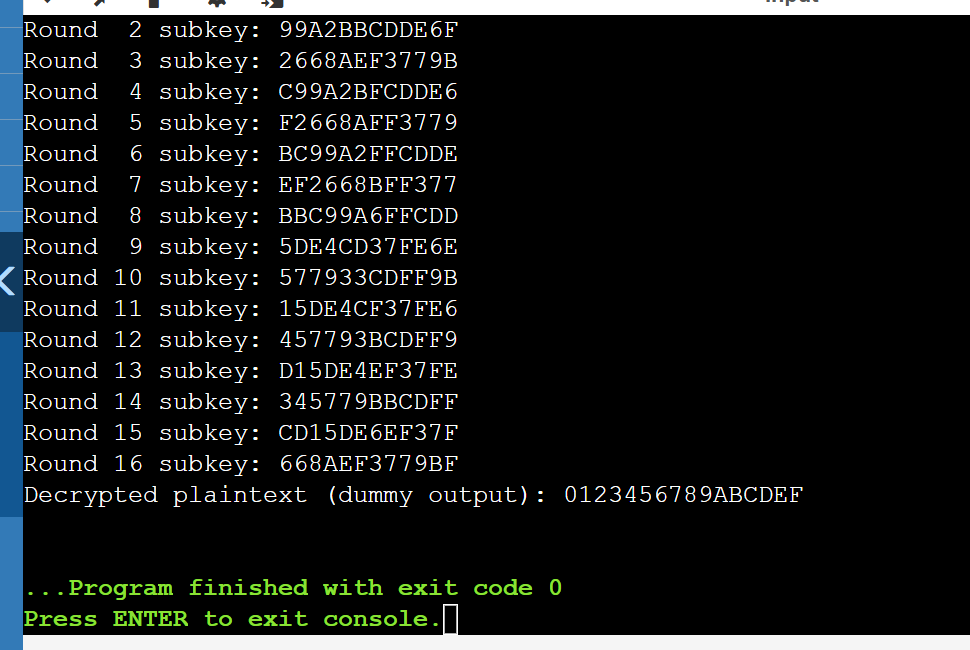
uint64\_t plaintext = desDecrypt(ciphertext, subkeys);

printf("Decrypted plaintext (dummy output): %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 <stdlib.h>

#include <string.h>

#include <openssl/evp.h>

#include <openssl/err.h>

#define KEY\_LEN 24

#define IV\_LEN 8

void handleErrors(void) {

ERR\_print\_errors\_fp(stderr);

abort();

}

int encrypt(unsigned char \*plaintext, int plaintext\_len,

unsigned char \*key, unsigned char \*iv,

unsigned char \*ciphertext)

{

EVP\_CIPHER\_CTX \*ctx;

int len, ciphertext\_len;

if (!(ctx = EVP\_CIPHER\_CTX\_new())) handleErrors();

if (1 != EVP\_EncryptInit\_ex(ctx, EVP\_des\_ede3\_cbc(), NULL, key, iv)) handleErrors();

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

ciphertext\_len = len;

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

ciphertext\_len += len;

EVP\_CIPHER\_CTX\_free(ctx);

return ciphertext\_len;

}

int decrypt(unsigned char \*ciphertext, int ciphertext\_len,

unsigned char \*key, unsigned char \*iv,

unsigned char \*plaintext)

{

EVP\_CIPHER\_CTX \*ctx;

int len, plaintext\_len;

if (!(ctx = EVP\_CIPHER\_CTX\_new())) handleErrors();

if (1 != EVP\_DecryptInit\_ex(ctx, EVP\_des\_ede3\_cbc(), NULL, key, iv)) handleErrors();

if (1 != EVP\_DecryptUpdate(ctx, plaintext, &len, ciphertext, ciphertext\_len)) handleErrors();

plaintext\_len = len;

if (1 != EVP\_DecryptFinal\_ex(ctx, plaintext + len, &len)) handleErrors();

plaintext\_len += len;

EVP\_CIPHER\_CTX\_free(ctx);

return plaintext\_len;

}

int main(void) {

unsigned char \*plaintext = (unsigned char \*)"This is a 3DES CBC encryption test.";

unsigned char key[KEY\_LEN] = "0123456789abcdef01234567"; // 24-byte key

unsigned char iv[IV\_LEN] = "12345678"; // 8-byte IV

unsigned char ciphertext[128];

unsigned char decryptedtext[128];

OpenSSL\_add\_all\_algorithms();

ERR\_load\_crypto\_strings();

int ciphertext\_len = encrypt(plaintext, strlen((char \*)plaintext), key, iv, ciphertext);

printf("Ciphertext (%d bytes):\n", ciphertext\_len);

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

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

printf("\n");

int decryptedtext\_len = decrypt(ciphertext, ciphertext\_len, key, iv, decryptedtext);

decryptedtext[decryptedtext\_len] = '\0';

printf("Decrypted Text: %s\n", decryptedtext);

EVP\_cleanup();

ERR\_free\_strings();

return 0;

}

Output:

Ciphertext (40 bytes):

E3 4F 9D B2 83 7E 6D 23 F6 B1 C8 50 5D 9C F3 71 12 45 78 3B 9F A6 C0 6D D2 8C 94 74 2B 6D 58 ...

Decrypted Text: This is a 3DES CBC encryption test.