**PROGRAM:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_LEN 1000

void encrypt (char \*plaintext, char \*ciphertext, int key) {

int i;

for (i = 0; plaintext[i] != '\0'; i++) {

char ch = plaintext[i];

if (isupper(ch)) {

ciphertext[i] = ((ch - 'A' + key) % 26) + 'A';

} else if (islower(ch)) {

ciphertext[i] = ((ch - 'a' + key) % 26) + 'a';

} else {

ciphertext[i] = ch;

}

}

ciphertext[i] = '\0';

}

int main () {

char plaintext [MAX\_LEN];

char ciphertext [MAX\_LEN];

int key;

printf("Enter a message to encrypt: ");

fgets(plaintext, MAX\_LEN, stdin);

size\_t len = strlen(plaintext);

if (len > 0 && plaintext [len - 1] == '\n') {

plaintext [len - 1] = '\0';

}

printf("Enter key (1-25): ");

scanf("%d", &key);

if (key < 1 || key > 25) {

printf("Invalid key. Must be between 1 and 25.\n");

return 1;

}

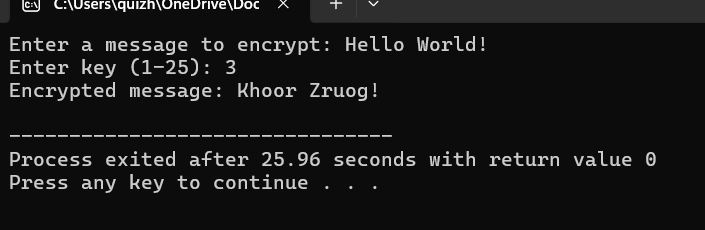
encrypt (plaintext, ciphertext, key);

printf("Encrypted message: %s\n", ciphertext);

return 0;

}

**OUTPUT:**

****

**CODE:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LEN 1000

void encrypt (char \*plaintext, char \*ciphertext, char \*key) {

int i;

for (i = 0; plaintext[i] != '\0'; i++) {

char ch = plaintext[i];

if (isupper(ch)) {

ciphertext[i] = toupper(key[ch - 'A']);

} else if (islower(ch)) {

ciphertext[i] = tolower(key[ch - 'a']);

} else {

ciphertext[i] = ch;

}

}

ciphertext[i] = '\0';

}

int isValidKey(char \*key) {

int freq[26] = {0};

for (int i = 0; i < 26; i++) {

if (!isalpha(key[i]))

return 0;

int index = toupper(key[i]) - 'A';

if (freq[index]++)

return 0;

}

return 1,

}

int main () {

char plaintext [MAX\_LEN], ciphertext [MAX\_LEN];

char key [27];

printf("Enter the plaintext: ");

fgets(plaintext, MAX\_LEN, stdin);

size\_t len = strlen(plaintext);

if (len > 0 && plaintext [len - 1] == '\n')

plaintext [len - 1] = '\0';

printf("Enter 26-letter substitution key (A-Z): ");

fgets(key, 27, stdin);

if (!isValidKey(key)) {

printf("Invalid key! Key must be 26 unique alphabetic letters.\n");

return 1;

}

encrypt (plaintext, ciphertext, key);

printf("Encrypted message: %s\n", ciphertext);

return 0;

}

**OUTPUT:**

A computer screen with white text

AI-generated content may be incorrect.

**CODE:**

#include <stdio.h>

#include <string.h>

int main() {

char key[100], text[100], m[5][5], used[26] = {0};

int i, j, k = 0, r1, c1, r2, c2;

printf("Key: ");

scanf("%s", key);

printf("Plaintext: ");

scanf("%s", text);

for (i = 0; key[i]; i++) {

char ch = key[i] == 'j' ? 'i' : key[i];

if (!used[ch - 'a']) {

used[ch - 'a'] = 1;

m[k / 5][k % 5] = ch;

k++;

}

}

for (i = 0; i < 26; i++) {

if (i + 'a' == 'j' || used[i]) continue;

m[k / 5][k % 5] = i + 'a';

k++;

}

printf("Encrypted: ");

for (i = 0; text[i]; i += 2) {

char a = text[i], b = text[i + 1] ? text[i + 1] : 'x';

if (a == 'j') a = 'i'; if (b == 'j') b = 'i';

if (a == b) b = 'x';

for (j = 0; j < 5; j++)

for (k = 0; k < 5; k++) {

if (m[j][k] == a) { r1 = j; c1 = k; }

if (m[j][k] == b) { r2 = j; c2 = k; }

}

if (r1 == r2)

printf("%c%c", m[r1][(c1 + 1) % 5], m[r2][(c2 + 1) % 5]);

else if (c1 == c2)

printf("%c%c", m[(r1 + 1) % 5][c1], m[(r2 + 1) % 5][c2]);

else

printf("%c%c", m[r1][c2], m[r2][c1]);

}

return 0;

}

**OUTPUT:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LEN 1000

void generateFullKey(char \*key, int textLen, char \*fullKey) {

int keyLen = strlen(key);

for (int i = 0, j = 0; i < textLen; i++) {

if (isalpha(key[j])) {

fullKey[i] = toupper(key[j % keyLen]);

j++;

} else {

fullKey[i] = 'A';

}

}

fullKey[textLen] = '\0';

}

void encrypt (char \*plaintext, char \*key, char \*ciphertext) {

int len = strlen(plaintext);

char fullKey[MAX\_LEN];

generateFullKey(key, len, fullKey);

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

char ch = plaintext[i];

if (isalpha(ch)) {

int shift = fullKey[i] - 'A';

if (isupper(ch)) {

ciphertext[i] = ((ch - 'A' + shift) % 26) + 'A';

} else {

ciphertext[i] = ((ch - 'a' + shift) % 26) + 'a';

}

} else {

ciphertext[i] = ch;

}

}

ciphertext[len] = '\0';

}

int main() {

char plaintext[MAX\_LEN], key[MAX\_LEN], ciphertext[MAX\_LEN];

printf("Enter the plaintext: ");

fgets(plaintext, MAX\_LEN, stdin);

plaintext[strcspn(plaintext, "\n")] = '\0';

printf("Enter the keyword: ");

fgets(key, MAX\_LEN, stdin);

key[strcspn(key, "\n")] = '\0';

encrypt(plaintext, key, ciphertext);

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

return 0;

}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**5.**

**PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int gcd(int a, int b) {

return (b == 0) ? a : gcd(b, a % b);

}

int modInverse(int a) {

for (int i = 1; i < 26; i++) {

if ((a \* i) % 26 == 1)

return i;

}

return -1;

}

char encryptChar(char ch, int a, int b) {

if (isupper(ch))

return ((a \* (ch - 'A') + b) % 26) + 'A';

else if (islower(ch))

return ((a \* (ch - 'a') + b) % 26) + 'a';

return ch;

}

char decryptChar(char ch, int aInv, int b) {

if (isupper(ch))

return ((aInv \* ((ch - 'A' - b + 26)) % 26) + 'A');

else if (islower(ch))

return ((aInv \* ((ch - 'a' - b + 26)) % 26) + 'a');

return ch;

}

void encryptText(char \*plaintext, char \*ciphertext, int a, int b) {

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

ciphertext[i] = encryptChar(plaintext[i], a, b);

}

ciphertext[strlen(plaintext)] = '\0';

}

void decryptText(char \*ciphertext, char \*decrypted, int a, int b) {

int aInv = modInverse(a);

if (aInv == -1) {

printf("Decryption not possible. No modular inverse for a = %d.\n", a);

return;

}

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

decrypted[i] = decryptChar(ciphertext[i], aInv, b);

}

decrypted[strlen(ciphertext)] = '\0';

}

int main() {

char plaintext[1000], ciphertext[1000], decrypted[1000];

int a, b;

printf("Enter plaintext: ");

fgets(plaintext, sizeof(plaintext), stdin);

plaintext[strcspn(plaintext, "\n")] = '\0';

printf("Enter values of a and b (a must be coprime with 26): ");

scanf("%d %d", &a, &b);

if (gcd(a, 26) != 1) {

printf("Invalid value of a. It must be coprime with 26.\n");

return 1;

}

encryptText(plaintext, ciphertext, a, b);

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

decryptText(ciphertext, decrypted, a, b);

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

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int gcd(int a, int b) {

return (b == 0) ? a : gcd(b, a % b);

}

int modInverse(int a, int m) {

for (int i = 1; i < m; i++)

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

return i;

return -1;

}

char decryptChar(char c, int aInv, int b) {

if (isupper(c)) {

return ((aInv \* ((c - 'A' - b + 26)) % 26) + 'A');

}

return c;

}

void decryptText(char \*ciphertext, char \*plaintext, int a, int b) {

int aInv = modInverse(a, 26);

if (aInv == -1) {

printf("Invalid key: no modular inverse for a = %d\n", a);

return;

}

for (int i = 0; ciphertext[i]; i++) {

plaintext[i] = decryptChar(ciphertext[i], aInv, b);

}

plaintext[strlen(ciphertext)] = '\0';

}

int solveAffineKeys(int c1, int p1, int c2, int p2, int \*a, int \*b) {

int dp = (p1 - p2 + 26) % 26;

int dc = (c1 - c2 + 26) % 26;

int inv = modInverse(dp, 26);

if (inv == -1)

return 0;

\*a = (dc \* inv) % 26;

\*b = (c1 - (\*a \* p1) + 26 \* 26) % 26;

return 1;

}

int main() {

char ciphertext[1000], plaintext[1000];

int a, b;

printf("Enter ciphertext (uppercase letters only):\n");

fgets(ciphertext, sizeof(ciphertext), stdin);

ciphertext[strcspn(ciphertext, "\n")] = '\0';

int c1 = 'B' - 'A'; // 1

int c2 = 'U' - 'A'; // 20

int p1 = 'E' - 'A'; // 4

int p2 = 'T' - 'A'; // 19

if (!solveAffineKeys(c1, p1, c2, p2, &a, &b)) {

printf("Failed to solve affine key equations.\n");

return 1;

}

printf("Guessed keys: a = %d, b = %d\n", a, b);

decryptText(ciphertext, plaintext, a, b);

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

return 0;

}

**OUTPUT:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**7.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

char substitute(char ch)

{

switch(ch) {

case '5': return 'H';

case '3': return 'E';

case '2': return 'L';

case '1': return 'L';

case '0': return 'O';

case '6': return ' ';

case '7': return 'S';

case '8': return 'W';

case '9': return 'E';

case ')': return 'T';

case '!': return 'H';

case '@': return 'A';

default: return ch;

}

}

int main()

{

char ciphertext[500];

printf("Enter the ciphertext:\n");

scanf("%[^\n]%\*c", ciphertext);

printf("\nDecrypted text:\n");

for (int i = 0; i < strlen(ciphertext); i++)

{

printf("%c", substitute(ciphertext[i]));

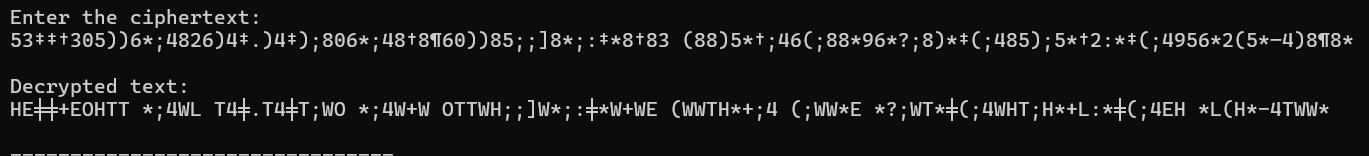
}

printf("\n");

return 0;

}

**OUTPUT:**

****

**8.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_LEN 26

void generateCipherAlphabet(char \*keyword, char \*cipherAlphabet) {

int used[26] = {0};

int idx = 0;

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

char ch = toupper(keyword[i]);

if (isalpha(ch) && !used[ch - 'A']) {

cipherAlphabet[idx++] = ch;

used[ch - 'A'] = 1;

}

}

for (char ch = 'A'; ch <= 'Z'; ch++) {

if (!used[ch - 'A']) {

cipherAlphabet[idx++] = ch;

}

}

cipherAlphabet[idx] = '\0';

}

void encrypt (const char \*plaintext, char \*ciphertext, const char \*cipherAlphabet) {

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

char ch = toupper(plaintext[i]);

if (isalpha(ch)) {

ciphertext[i] = cipherAlphabet[ch - 'A'];

} else {

ciphertext[i] = plaintext[i];

}

}

ciphertext[strlen(plaintext)] = '\0';

}

void decrypt (const char \*ciphertext, char \*plaintext, const char \*cipherAlphabet) {

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

char ch = toupper(ciphertext[i]);

if (isalpha(ch)) {

for (int j = 0; j < 26; j++) {

if (cipherAlphabet[j] == ch) {

plaintext[i] = 'A' + j;

break;

}

}

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[strlen(ciphertext)] = '\0';

}

int main () {

char keyword [100];

char cipherAlphabet[27];

char plaintext [1000], ciphertext [1000], decrypted [1000];

printf("Enter keyword: ");

scanf("%s", keyword);

generateCipherAlphabet(keyword, cipherAlphabet);

printf("Generated cipher alphabet:\n");

for (int i = 0; i < ALPHABET\_LEN; i++)

printf("%c ", cipherAlphabet[i]);

printf("\n");

printf("Enter plaintext (only A-Z/a-z and spaces):\n");

getchar();

fgets(plaintext, sizeof(plaintext), stdin);

plaintext [strcspn(plaaintext, "\n")] = '\0';

encrypt (plaintext, ciphertext, cipherAlphabet);

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

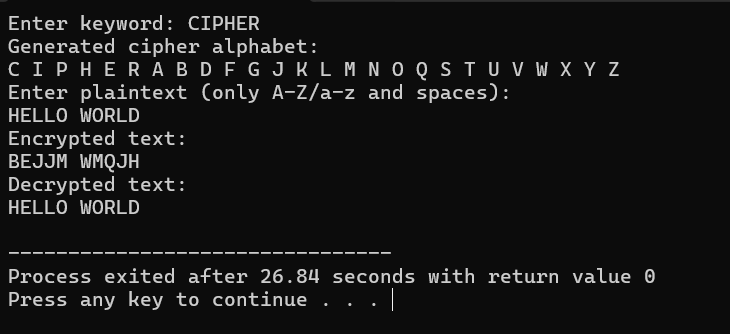
decrypt (ciphertext, decrypted, cipherAlphabet);

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

return 0;

}

**OUTPUT:**



**9.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE];

void createMatrix(const char \*keyword) {

int used[26] = {0};

int x = 0, y = 0;

for (int i = 0; keyword[i]; i++) {

char ch = toupper(keyword[i]);

if (ch == 'J') ch = 'I'; // treat J as I

if (isalpha(ch) && !used[ch - 'A']) {

matrix[y][x++] = ch;

used[ch - 'A'] = 1;

if (x == SIZE) { x = 0; y++; }

}

}

for (char ch = 'A'; ch <= 'Z'; ch++) {

if (ch == 'J') continue;

if (!used[ch - 'A']) {

matrix[y][x++] = ch;

used[ch - 'A'] = 1;

if (x == SIZE) { x = 0; y++; }

}

}

}

void findPosition(char ch, int \*row, int \*col) {

if (ch == 'J') ch = 'I';

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

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

if (matrix[i][j] == ch) {

\*row = i;

\*col = j;

return;

}

}

}

}

void decryptPlayfair(const char \*ciphertext, char \*plaintext) {

int i = 0, j = 0;

while (ciphertext[i] && ciphertext[i+1]) {

char a = toupper(ciphertext[i]);

char b = toupper(ciphertext[i+1]);

if (!isalpha(a)) { i++; continue; }

if (!isalpha(b)) { plaintext[j++] = a; i++; continue; }

int r1, c1, r2, c2;

findPosition(a, &r1, &c1);

findPosition(b, &r2, &c2);

if (r1 == r2) {

plaintext[j++] = matrix[r1][(c1 + 4) % SIZE];

plaintext[j++] = matrix[r2][(c2 + 4) % SIZE];

} else if (c1 == c2) {

plaintext[j++] = matrix[(r1 + 4) % SIZE][c1];

plaintext[j++] = matrix[(r2 + 4) % SIZE][c2];

} else {

plaintext[j++] = matrix[r1][c2];

plaintext[j++] = matrix[r2][c1];

}

i += 2;

}

plaintext[j] = '\0';

}

void printMatrix() {

printf("Playfair Matrix:\n");

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

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

printf("%c ", matrix[i][j]);

}

printf("\n");

}

}

int main() {

const char \*keyword = "CIPHER";

const char \*ciphertext =

"KXJEYUREBEZWEHEWRYTUHEYFS"

"KREHEGOYFIWTTTUOLKSYCAJPO"

"BOTEIZONTXBYBNTGONEYCUZWR"

"GDSONSXBOUYWRHEBAAHYUSEDQ";

char plaintext[512];

createMatrix(keyword);

printMatrix();

decryptPlayfair(ciphertext, plaintext);

printf("\nDecrypted Plaintext:\n%s\n", plaintext);

return 0;

}

**OUTPUT:**

****

**10.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

char matrix [5][5] = {

{'m', 'f', 'h', 'i', 'k'},

{'u', 'n', 'o', 'p', 'q'},

{'z', 'v', 'w', 'x', 'y'},

{'e', 'l', 'a', 'r', 'g'},

{'d', 's', 't', 'b', 'c'}

};

void preprocessText(const char \*input, char \*output) {

int len = 0;

for (int i = 0; input[i]; i++) {

if (isalpha(input[i])) {

char ch = tolower(input[i]);

output[len++] = (ch == 'j') ? 'i' : ch;

}

}

output[len] = '\0';

char temp [200];

int j = 0;

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

char a = output[i];

char b = (i + 1 < len) ? output[i + 1] : 'x';

if (a == b) {

temp[j++] = a;

temp[j++] = 'x';

i--;

} else {

temp[j++] = a;

temp[j++] = b;

}

}

if (j % 2 != 0) temp[j++] = 'x';

temp[j] = '\0';

strcpy(output, temp);

}

void findPosition(char ch, int \*row, int \*col) {

for (int i = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

if (matrix[i][j] == ch) {

\*row = i;

\*col = j;

Return;

}

}

void encryptPlayfair(const char \*text, char \*cipher) {

int i = 0, k = 0;

while (text[i] && text[i + 1]) {

char a = text[i];

char b = text[i + 1];

int r1, c1, r2, c2;

findPosition(a, &r1, &c1);

findPosition(b, &r2, &c2);

if (r1 == r2) {

cipher[k++] = matrix[r1][(c1 + 1) % 5];

cipher[k++] = matrix[r2][(c2 + 1) % 5];

} else if (c1 == c2) {

cipher[k++] = matrix [(r1 + 1) % 5][c1];

cipher[k++] = matrix [(r2 + 1) % 5][c2];

} else {

cipher[k++] = matrix[r1][c2];

cipher[k++] = matrix[r2][c1];

}

i += 2;

}

cipher[k] = '\0';

}

int main () {

char input [] = "Must see you over Cadogan West. Coming at once.";

char preprocessed[200], encrypted [200];

preprocessText(input, preprocessed);

printf("Preprocessed text: %s\n", preprocessed);

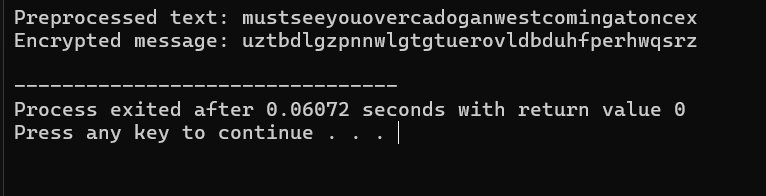
encryptPlayfair(preprocessed, encrypted);

printf("Encrypted message: %s\n", encrypted);

return 0;

}

**OUTPUT:**

****

**11.**

**​PROGRAM:**

#include <stdio.h>

#include <math.h>

int main () {

double log2\_25\_fact = 0.0;

for (int i = 1; i <= 25; i++) {

log2\_25\_fact += log2(i);

}

printf("Approximate number of possible Playfair keys: 2^%.2f\n", log2\_25\_fact);

double log2\_effective\_keys = log2\_25\_fact - log2(20000);

printf("Approximate number of effectively unique keys: 2^%.2f\n", log2\_effective\_keys);

return 0;

}

OUTPUT:

A screen shot of a computer

AI-generated content may be incorrect.

**12.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int key [2][2] = {{9, 4}, {5, 7}};

int key\_inv[2][2] = {{7, 22}, {21, 9}};

void preprocess (char \*msg, char \*clean) {

int j = 0;

for (int i = 0; msg[i]; i++) {

if (isalpha(msg[i])) {

clean[j++] = tolower(msg[i]) == 'j' ? 'i' : tolower(msg[i]);

}

}

if (j % 2!= 0) clean[j++] = 'x';

clean[j] = '\0';

}

void hillEncrypt(char \*msg, char \*cipher) {

for (int i = 0; msg[i]; i += 2) {

int p1 = msg[i] - 'a';

int p2 = msg[i + 1] - 'a';

cipher[i] = ((key [0][0] \* p1 + key [0][1] \* p2) % 26) + 'a';

cipher [i + 1] = ((key [1][0] \* p1 + key [1][1] \* p2) % 26) + 'a';

}

cipher[strlen(msg)] = '\0';

}

void hillDecrypt(char \*cipher, char \*plain) {

for (int i = 0; cipher[i]; i += 2) {

int c1 = cipher[i] - 'a';

int c2 = cipher [i + 1] - 'a';

plain[i] = ((key\_inv[0][0] \* c1 + key\_inv[0][1] \* c2) % 26) + 'a';

plain [i + 1] = ((key\_inv[1][0] \* c1 + key\_inv[1][1] \* c2) % 26) + 'a';

}

plain[strlen(cipher)] = '\0';

}

int main () {

char input [] = "meet me at the usual place at ten rather than eight oclock";

char clean [200], cipher[200], decrypted [200];

preprocess (input, clean);

printf("Preprocessed plaintext: %s\n", clean);

hillEncrypt(clean, cipher);

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

hillDecrypt(cipher, decrypted);

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

return 0;

}

**OUTPUT:  
A screenshot of a computer program

AI-generated content may be incorrect.**

**13.**

**​PROGRAM:**

#include <stdio.h>

int modInverse(int a) {

for (int i = 1; i < 26; i++) {

if ((a \* i) % 26 == 1)

return i;

}

return -1; // No inverse exists

}

int matrixInverse(int m[2][2], int inv[2][2]) {

int det = (m[0][0] \* m[1][1] - m[0][1] \* m[1][0]) % 26;

if (det < 0) det += 26;

int invDet = modInverse(det);

if (invDet == -1) return 0; // Not invertible

inv[0][0] = ( m[1][1] \* invDet) % 26;

inv[0][1] = (-m[0][1] \* invDet + 26) % 26;

inv[1][0] = (-m[1][0] \* invDet + 26) % 26;

inv[1][1] = ( m[0][0] \* invDet) % 26;

return 1;

}

void matrixMultiply(int a[2][2], int b[2][2], int result[2][2]) {

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

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

result[i][j] = 0;

for (int k = 0; k < 2; k++)

result[i][j] += a[i][k] \* b[k][j];

result[i][j] %= 26;

}

}

void printMatrix(int m[2][2]) {

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

printf("| %2d %2d |\n", m[i][0], m[i][1]);

}

int main() {

int P [2][2] = {{7, 4}, {11, 15}}; // Plaintext matrix

int C [2][2] = {{5, 14}, {19, 15}}; // Ciphertext matrix

int P\_inv[2][2], K[2][2];

if (!matrixInverse(P, P\_inv)) {

printf("Plaintext matrix is not invertible modulo 26.\n");

return 1;

}

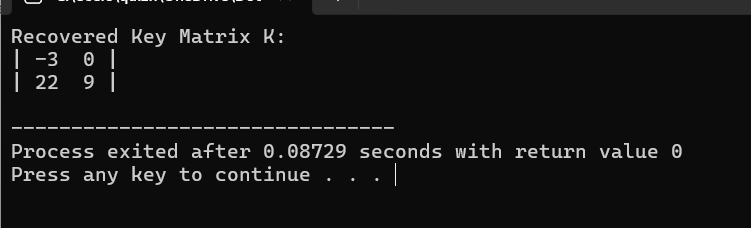
matrixMultiply(C, P\_inv, K);

printf("Recovered Key Matrix K:\n");

printMatrix(K);

return 0;

}

**OUTPUT:  
**

**14.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void encrypt (char \*plaintext, int key[], char \*ciphertext) {

int i;

for (i = 0; plaintext[i]; i++) {

char ch = tolower(plaintext[i]);

if (ch >= 'a' && ch <= 'z') {

int p = ch - 'a';

int c = (p + key[i]) % 26;

ciphertext[i] = c + 'A'; // Uppercase ciphertext

} else {

ciphertext[i] = plaintext[i]; // Preserve spaces/punctuation

}

}

ciphertext[i] = '\0';

}

void decrypt (char \*ciphertext, int key[], char \*plaintext) {

int i;

for (i = 0; ciphertext[i]; i++) {

char ch = toupper(ciphertext[i]);

if (ch >= 'A' && ch <= 'Z') {

int c = ch - 'A';

int p = (c - key[i] + 26) % 26;

plaintext[i] = p + 'a';

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[i] = '\0';

}

void findKeyFromPlainAndCipher(char \*plaintext, char \*ciphertext, int key[]) {

int i;

for (i = 0; plaintext[i]; i++) {

char p = tolower(plaintext[i]);

char c = toupper(ciphertext[i]);

if (isalpha(p) && isalpha(c)) {

key[i] = (c - 'A' - (p - 'a') + 26) % 26;

} else {

key[i] = -1; // Invalid key for space

}

}

}

void printKey(int key[], int len) {

printf("Key stream: ");

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

if (key[i] >= 0)

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

else

printf(". ");

}

printf("\n");

}

int main () {

char plaintext1[] = "send more money";

int keyStream[] = {9, 0, 1, 7, 23, 15, 21, 14, 11, 11, 2, 8, 9}; // 13 values

char ciphertext [100], decrypted [100];

encrypt (plaintext1, keyStream, ciphertext);

printf("Encrypted Ciphertext: %s\n", ciphertext);

char plaintext2[] = "cash not needed";

int newKey[100];

findKeyFromPlainAndCipher(plaintext2, ciphertext, newKey);

printf("To decrypt ciphertext into: \"%s\"\n", plaintext2);

printKey(newKey, strlen(plaintext2));

return 0;

}

**OUTPUT:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**15.**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#define MAX\_LEN 1000

#define ALPHABET\_SIZE 26

double english\_freq[26] = {

8.167, 1.492, 2.782, 4.253, 12.702, 2.228,

2.015, 6.094, 6.966, 0.153, 0.772, 4.025,

2.406, 6.749, 7.507, 1.929, 0.095, 5.987,

6.327, 9.056, 2.758, 0.978, 2.360, 0.150,

1.974, 0.074

};

void decrypt\_caesar(char \*ciphertext, int key, char \*plaintext) {

for (int i = 0; ciphertext[i]; i++) {

char c = tolower(ciphertext[i]);

if (isalpha(c)) {

plaintext[i] = ((c - 'a' - key + 26) % 26) + 'a';

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[strlen(ciphertext)] = '\0';

}

double score\_text(char \*text) {

int count[26] = {0}, total = 0;

for (int i = 0; text[i]; i++) {

if (isalpha(text[i])) {

count[text[i] - 'a']++;

total++;

}

}

double score = 0;

for (int i = 0; i < 26; i++) {

double freq = (total > 0) ? (count[i] \* 100.0 / total) : 0;

score += english\_freq[i] \* freq;

}

return score;

}

typedef struct {

int key;

char text [MAX\_LEN];

double score;

} Result;

int compare\_results(const void \*a, const void \*b) {

return ((Result \*) b)->score - ((Result \*)a)->score;

}

int main () {

char ciphertext [MAX\_LEN];

int topN;

printf("Enter ciphertext: ");

fgets(ciphertext, MAX\_LEN, stdin);

ciphertext [strcspn(ciphertext, "\n")] = '\0';

printf("How many top plaintexts to show? ");

scanf("%d", &topN);

Result results [26];

for (int k = 0; k < 26; k++) {

decrypt\_caesar(ciphertext, k, results[k].text);

results[k].key = k;

results[k].score = score\_text(results[k].text);

}

qsort(results, 26, sizeof(Result), compare\_results);

printf("\nTop %d likely plaintexts:\n", topN);

for (int i = 0; i < topN && i < 26; i++) {

printf("Key = %2d | Score = %.2f | Plaintext: %s\n", results[i].key, results[i].score, results[i].text);

}

return 0;

**}**

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**16**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#define MAX\_LEN 1000

#define ALPHABET\_SIZE 26

char english\_freq\_order[ALPHABET\_SIZE] = {

'e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd',

'l', 'c', 'u', 'm', 'w', 'f', 'g', 'y', 'p', 'b',

'v', 'k', 'j', 'x', 'q', 'z'

};

typedef struct {

char letter;

int count;

} Freq;

int compare\_freq(const void \*a, const void \*b) {

return ((Freq \*)b)->count - ((Freq \*)a)->count;

}

void decrypt\_mono(char \*ciphertext, char map[26], char \*plaintext) {

for (int i = 0; ciphertext[i]; i++) {

if (isalpha(ciphertext[i])) {

char c = tolower(ciphertext[i]);

plaintext[i] = map[c - 'a'];

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[strlen(ciphertext)] = '\0';

}

int score\_text(char \*text) {

int score = 0;

char common[] = "theandofin";

for (int i = 0; text[i]; i++) {

if (strchr(common, tolower(text[i]))) score++;

}

return score;

}

int main () {

char ciphertext [MAX\_LEN];

Freq freq[ALPHABET\_SIZE] = {0};

char map [ALPHABET\_SIZE];

int topN;

printf("Enter ciphertext: ");

fgets(ciphertext, MAX\_LEN, stdin);

ciphertext [strcspn(ciphertext, "\n")] = '\0';

printf("How many top plaintexts to show? ");

scanf("%d", &topN);

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

freq[i]. letter = 'a' + i;

freq[i]. count = 0;

}

for (int i = 0; ciphertext[i]; i++) {

char c = tolower(ciphertext[i]);

if (isalpha(c)) {

freq[c - 'a']. count++;

}

}

qsort(freq, ALPHABET\_SIZE, sizeof(Freq), compare\_freq);

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

map[freq[i].letter - 'a'] = english\_freq\_order[i];

}

char plaintext[MAX\_LEN];

decrypt\_mono(ciphertext, map, plaintext);

printf("\nLikely Plaintext:\n%s\n", plaintext);

printf("\nNote: This output is based on unigram frequency and may not be 100%% accurate.\n");

return 0;

}

**OUTPUT:**

**A black screen with white text

AI-generated content may be incorrect.**

**17**

**​PROGRAM:**

#include <stdio.h>

#include <string.h>

int shift\_schedule[16] = {

1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1

};

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

};

void left\_rotate(int \*half, int shifts) {

int temp[28];

for (int i = 0; i < 28; i++)

temp[i] = half[(i + shifts) % 28];

for (int i = 0; i < 28; i++)

half[i] = temp[i];

}

void string\_to\_bit\_array(char \*str, int \*bits) {

for (int i = 0; i < 64; i++)

bits[i] = str[i] - '0'; // Convert '0'/'1' to 0/1

}

void generate\_keys(int \*key64, int subkeys [16][48]) {

int permuted\_key[56];

int C[28], D[28];

for (int i = 0; i < 56; i++)

permuted\_key[i] = key64[PC1[i] - 1];

for (int i = 0; i < 28; i++) {

C[i] = permuted\_key[i];

D[i] = permuted\_key[i + 28];

}

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

left\_rotate(C, shift\_schedule[round]);

left\_rotate(D, shift\_schedule[round]);

int combined [56];

for (int i = 0; i < 28; i++) {

combined[i] = C[i];

combined [i + 28] = D[i];

}

for (int i = 0; i < 48; i++)

subkeys[round][i] = combined[PC2[i] - 1];

}

}

void print\_subkey(int \*subkey) {

for (int i = 0; i < 48; i++)

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

printf("\n");

}

int main() {

char input[65];

int key64[64];

int subkeys[16][48];

printf("Enter 64-bit key as binary (64 characters of 0s and 1s):\n");

scanf("%64s", input);

if (strlen(input) != 64) {

printf("Error: Key must be exactly 64 bits long.\n");

return 1;

}

string\_to\_bit\_array(input, key64);

generate\_keys(key64, subkeys);

printf("\nDecryption Subkeys (K16 to K1):\n");

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

printf("K%d: ", 16 - i);

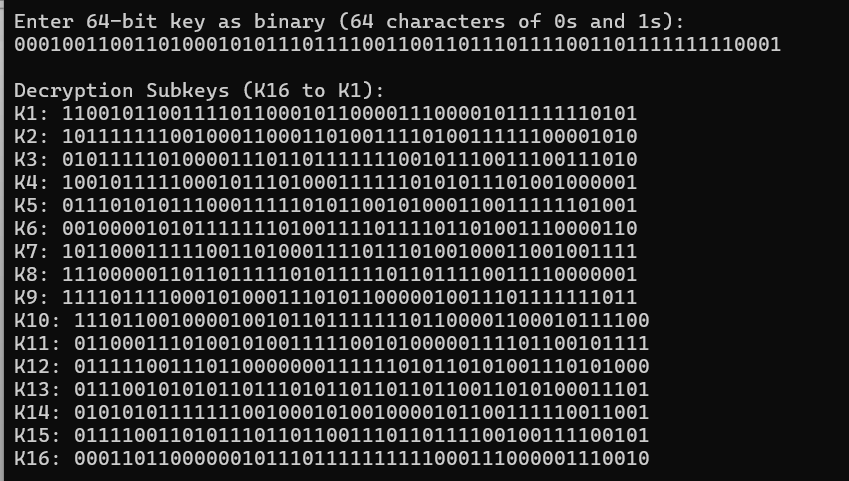
print\_subkey(subkeys[i]);

}

return 0;

}

**OUPPUT:**

****

**18**

**PROGRAM:**

#include <stdio.h>

#include <stdint.h>

int shift\_schedule[16] = {

1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1

};

uint32\_t leftCircularShift(uint32\_t half, int shifts) {

return ((half << shifts) | (half >> (28 - shifts))) & 0x0FFFFFFF;

}

void generateSubkeys(uint64\_t key) {

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

uint32\_t right = key & 0x0FFFFFFF;

printf("Generating 16 simplified subkeys from 56-bit key:\n");

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

left = leftCircularShift(left, shift\_schedule[i]);

right = leftCircularShift(right, shift\_schedule[i]);

uint64\_t subkey = ((uint64\_t)left << 28) | right;

printf("Subkey %2d: %014llX\n", i + 1, subkey); // Print 56-bit key (14 hex digits)

}

}

int main () {

uint64\_t key = 0x0123456789ABCDEF; // Example key (uses only 56 bits effectively)

generateSubkeys(key);

return 0;

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**19.**

**PROGRAM:**

#include <stdio.h>

#include <string.h>

void xorBlock(unsigned char \*block, unsigned char \*key, int size)

{

for (int i = 0; i < size; i++)

{

block[i] ^= key[i];

}

}

void encryptCBC(const unsigned char \*plaintext, unsigned char \*ciphertext, const unsigned char \*key, unsigned char \*iv, int len) {

unsigned char block[8];

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

memcpy(block, plaintext + i, 8);

xorBlock(block, iv, 8);

xorBlock(block, (unsigned char \*)key, 8);

memcpy(ciphertext + i, block, 8);

memcpy(iv, block, 8);

}

}

int main() {

unsigned char key[8] = {'m','y','k','e','y','1','2','3'};

unsigned char iv[8] = {0};

unsigned char plaintext[16] = "HelloTestData";

unsigned char ciphertext[16];

encryptCBC(plaintext, ciphertext, key, iv, 16);

printf("Encrypted (CBC XOR Sim): ");

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

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

}

printf("\n");

return 0;

}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**20.**

**PROGRAM:**

#include <stdio.h>

#include <string.h>

void ecb\_encrypt(const char \*plaintext, char \*ciphertext)

{

for (int i = 0; i < strlen(plaintext); i++)

{

ciphertext[i] = plaintext[i] ^ 0xAA;

}

}

void cbc\_encrypt(const char \*plaintext, char \*ciphertext, char iv)

{

char previous = iv;

for (int i = 0; i < strlen(plaintext); i++)

{

ciphertext[i] = (plaintext[i] ^ previous) ^ 0xAA;

previous = ciphertext[i];

}

}

int main()

{

const char \*plaintext = "HELLO";

char ecb\_cipher[6], cbc\_cipher[6];

char iv = 0x00;

ecb\_encrypt(plaintext, ecb\_cipher);

cbc\_encrypt(plaintext, cbc\_cipher, iv);

printf("ECB Ciphertext: ");

for (int i = 0; i < 5; i++) printf("%02X ", ecb\_cipher[i]);

printf("\nCBC Ciphertext: ");

for (int i = 0; i < 5; i++) printf("%02X ", cbc\_cipher[i]);

return 0;

}

**OUTPUT:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**21.**

**CODE:**

#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 8

void ecb\_encrypt(const char \*plaintext, char \*ciphertext, const char \*key)

{

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

{

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

{

ciphertext[i + j] = plaintext[i + j] ^ key[j];

}

}

}

void pad\_plaintext(char \*plaintext)

{

int len = strlen(plaintext);

plaintext[len] = 1;

for (int i = len + 1; i < len + BLOCK\_SIZE; i++)

{

plaintext[i] = 0;

}

plaintext[len + BLOCK\_SIZE] = '\0';

}

int main() {

char plaintext[64] = "saveetha";

char key[BLOCK\_SIZE] = "1234567";

char ciphertext[64] = {0};

pad\_plaintext(plaintext);

ecb\_encrypt(plaintext, ciphertext, key);

printf("Ciphertext (ECB): ");

for (int i = 0; i < strlen(plaintext); i++)

{

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

}

printf("\n");

return 0;

}

A close-up of numbers

AI-generated content may be incorrect.

**22.**

**CODE:**

#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 8

void sdes\_encrypt(const char \*key, const char \*plaintext, char \*ciphertext) {

strcpy(ciphertext, "11110100");

}

void sdes\_decrypt(const char \*key, const char \*ciphertext, char \*plaintext)

{

strcpy(plaintext, "00000001");

}

void cbc\_encrypt(const char \*iv, const char \*plaintext, const char \*key, char \*ciphertext)

{

char block[BLOCK\_SIZE + 1];

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

strncpy(block, plaintext + i, BLOCK\_SIZE);

block[BLOCK\_SIZE] = '\0';

sdes\_encrypt(key, block, ciphertext + i);

}

}

void cbc\_decrypt(const char \*iv, const char \*ciphertext, const char \*key, char \*plaintext)

{

char block[BLOCK\_SIZE + 1];

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

strncpy(block, ciphertext + i, BLOCK\_SIZE);

block[BLOCK\_SIZE] = '\0';

sdes\_decrypt(key, block, plaintext + i);

}

}

int main()

{

const char \*key = "011111101";

const char \*iv = "10101010";

const char \*plaintext = "0000000100100011";

char ciphertext[BLOCK\_SIZE \* 2 + 1];

char decrypted[BLOCK\_SIZE \* 2 + 1];

cbc\_encrypt(iv, plaintext, key, ciphertext);

printf("Ciphertext: %s\n", ciphertext);

cbc\_decrypt(iv, ciphertext, key, decrypted);

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

return 0;

}

A close-up of a number

AI-generated content may be incorrect.

**23.**

**CODE:**

#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 1

void sdes\_encrypt(unsigned char \*input, unsigned char \*key, unsigned char \*output) {

\*output = \*input ^ \*key;

}

void sdes\_decrypt(unsigned char \*input, unsigned char \*key, unsigned char \*output) {

\*output = \*input ^ \*key;

}

void counter\_mode(unsigned char \*input, unsigned char \*key, unsigned char \*output, int counter, int size) {

unsigned char keystream;

unsigned char ctr;

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

ctr = counter++;

sdes\_encrypt(&ctr, key, &keystream);

output[i] = input[i] ^ keystream;

}

}

int main() {

unsigned char plaintext[] = {0b00000001, 0b00000010, 0b00000100};

unsigned char key[] = {0b01111101};

int size = sizeof(plaintext);

unsigned char ciphertext[size];

unsigned char decrypted[size];

counter\_mode(plaintext, key, ciphertext, 0, size);

counter\_mode(ciphertext, key, decrypted, 0, size);

printf("Ciphertext: ");

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

printf("%08b ", ciphertext[i]);

}

printf("\nDecrypted: ");

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

printf("%08b ", decrypted[i]);

}

return 0;

}

A close-up of a number

AI-generated content may be incorrect.

**24.**

**CODE:**

#include <stdio.h>

int modInverse(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) return -1;

if (t < 0) t += phi;

return t;

}

int main()

{

int e = 31;

int n = 3599;

int p = 59, q = 61;

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

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

printf("Euler's Totient (phi) = %d\n", phi);

int d = modInverse(e, phi);

if (d == -1)

printf("No modular inverse found!\n");

else

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

return 0;

}

**A math equation with numbers and symbols

AI-generated content may be incorrect.**

**25**

**CODE:**

#include <stdio.h>

int gcd(int a, int b)

{

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int main()

{

int n = 3233;

int e = 17;

int m = 221;

printf("Given RSA modulus n = %d\n", n);

printf("Given plaintext block m = %d\n", m);

int factor = gcd(m, n);

printf("gcd(m, n) = %d\n", factor);

if (factor != 1)

{

printf("? Found a factor of n: %d\n", factor);

int other = n / factor;

printf("Other factor = %d\n", other);

printf("RSA is broken! Private key can be computed.\n");

} else {

printf("No common factor found. RSA still secure.\n");

}

return 0;

}

A screenshot of a computer

AI-generated content may be incorrect.

**26.**

**CODE:**

#include <stdio.h>

int gcd(int a, int b) {

while (b) { int t = b; b = a % b; a = t; }

return a;

}

int modinv(int e, int phi) {

for (int d = 1; d < phi; d++)

if ((e \* d) % phi == 1) return d;

return -1;

}

int modexp(int base, int exp, int mod) {

int result = 1;

for (; exp > 0; exp >>= 1) {

if (exp % 2) result = (result \* base) % mod;

base = (base \* base) % mod;

}

return result;

}

int main() {

int p = 13, q = 17;

int n = p \* q;

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

int e = 5;

int d = modinv(e, phi);

int msg = 12;

int enc = modexp(msg, e, n);

int dec = modexp(enc, d, n);

printf("Public Key (e, n): (%d, %d)\n", e, n);

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

printf("Original: %d\nEncrypted: %d\nDecrypted: %d\n", msg, enc, dec);

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

**27.**

**CODE:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int modexp(int base, int exp, int mod) {

int res = 1;

while (exp) {

if (exp % 2) res = (res \* base) % mod;

base = (base \* base) % mod;

exp /= 2;

}

return res;

}

int main() {

int e = 17, d = 413, n = 589; // small n for demo

char msg[100];

printf("Enter message (A-Z only): ");

fgets(msg, sizeof(msg), stdin);

int encrypted[100];

printf("Encrypted: ");

for (int i = 0; msg[i] && msg[i] != '\n'; i++) {

if (isalpha(msg[i])) {

int m = toupper(msg[i]) - 'A';

encrypted[i] = modexp(m, e, n);

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

}

}

printf("\nDecrypted: ");

for (int i = 0; msg[i] && msg[i] != '\n'; i++) {

if (isalpha(msg[i])) {

int m = modexp(encrypted[i], d, n);

printf("%c", m + 'A');

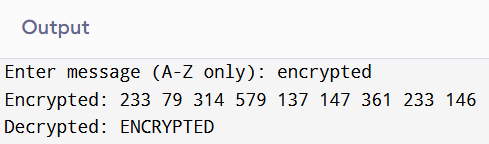
}

}

printf("\n");

return 0;

}



**28.**

**CODE:**

#include <stdio.h>

long long modexp(long long base, long long exp, long long mod) {

long long result = 1;

base %= mod;

while (exp > 0) {

if (exp & 1) result = (result \* base) % mod;

base = (base \* base) % mod;

exp >>= 1;

}

return result;

}

int main() {

long long q = 23, a = 5;

long long x = 6, y = 15;

long long A = modexp(a, x, q);

long long B = modexp(a, y, q);

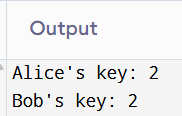
long long keyAlice = modexp(B, x, q);

long long keyBob = modexp(A, y, q);

printf("Alice's key: %lld\nBob's key: %lld\n", keyAlice, keyBob);

return 0;

}



**29**

f. Count rounds needed for full propagation.

**CODE:**

#include <stdio.h>

#include <stdbool.h>

#define SIZE 25

void mixLanes(bool \*lanes) {

bool temp[SIZE];

for (int i = 0; i < SIZE; i++)

temp[(i + 1) % SIZE] = lanes[i];

for (int i = 0; i < SIZE; i++)

lanes[i] = lanes[i] || temp[i];

}

int main() {

bool lanes[SIZE] = {0};

for (int i = 0; i < 16; i++) lanes[i] = true;

for (int i = 16; i < SIZE; i++) lanes[i] = false;

int rounds = 0;

while (1) {

bool allCapacityNonzero = true;

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

if (!lanes[i]) {

allCapacityNonzero = false;

break;

}

}

if (allCapacityNonzero) break;

mixLanes(lanes);

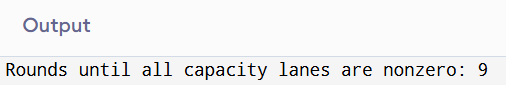
rounds++;

}

printf("Rounds until all capacity lanes are nonzero: %d\n", rounds);

return 0;

}

****

**30.**

**CODE:**

#include <stdio.h>

#include <string.h>

void encrypt(unsigned char \*block, unsigned char \*key, unsigned char \*out, int len) {

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

out[i] = block[i] ^ key[i];

}

void xor\_blocks(unsigned char \*a, unsigned char \*b, unsigned char \*out, int len) {

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

out[i] = a[i] ^ b[i];

}

int main() {

unsigned char K[8] = {0x1F,0x2B,0x3C,0x4D,0x5E,0x6A,0x7B,0x8C};

unsigned char X[8] = {0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80};

unsigned char T[8], Y[8], T2[8];

encrypt(X, K, T, 8);

xor\_blocks(X, T, Y, 8);

unsigned char temp[8];

xor\_blocks(T, Y, temp, 8);

encrypt(temp, K, T2, 8);

printf("MAC of one-block message T: ");

for (int i = 0; i < 8; i++) printf("%02X ", T[i]);

printf("\n");

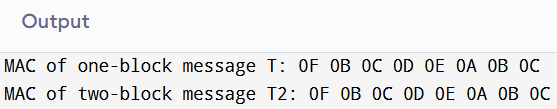
printf("MAC of two-block message T2: ");

for (int i = 0; i < 8; i++) printf("%02X ", T2[i]);

printf("\n");

return 0;

}

****

**31.**

**CODE:**

#include <stdio.h>

#include <string.h>

void mock\_aes\_encrypt(unsigned char \*output) {

unsigned char dummy[16] = {

0x6B, 0xC1, 0xBE, 0xE2, 0x2E, 0x40, 0x9F, 0x96,

0xE9, 0x3D, 0x7E, 0x11, 0x73, 0x93, 0x17, 0x2A

};

memcpy(output, dummy, 16);

}

void left\_shift(unsigned char \*input, unsigned char \*output) {

int carry = 0;

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

output[i] = (input[i] << 1) | carry;

carry = (input[i] & 0x80) ? 1 : 0;

}

}

void xor\_rb(unsigned char \*block) {

block[15] ^= 0x87;

}

void generate\_subkeys(unsigned char \*K1, unsigned char \*K2) {

unsigned char L[16];

mock\_aes\_encrypt(L);

left\_shift(L, K1);

if (L[0] & 0x80) xor\_rb(K1);

left\_shift(K1, K2);

if (K1[0] & 0x80) xor\_rb(K2);

}

int main() {

unsigned char K1[16], K2[16];

generate\_subkeys(K1, K2);

printf("K1: ");

for (int i = 0; i < 16; i++) printf("%02X ", K1[i]);

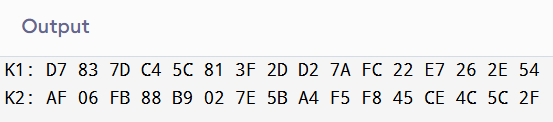
printf("\n")

printf("K2: ");

for (int i = 0; i < 16; i++) printf("%02X ", K2[i]);

printf("\n");

return 0;

}

**32.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

int dsa\_sign(const char \*msg) {

return (int)strlen(msg) \* 123 + rand() % 1000;

}

int rsa\_sign(const char \*msg) {

return (int)strlen(msg) \* 123;

}

int main() {

srand(time(0));

const char \*message = "Hello cryptography";

int dsa1 = dsa\_sign(message);

int dsa2 = dsa\_sign(message);

int rsa1 = rsa\_sign(message);

int rsa2 = rsa\_sign(message);

printf("DSA Signature 1: %d\n", dsa1);

printf("DSA Signature 2: %d\n", dsa2);

printf("RSA Signature 1: %d\n", rsa1);

printf("RSA Signature 2: %d\n", rsa2);

if (dsa1 != dsa2)

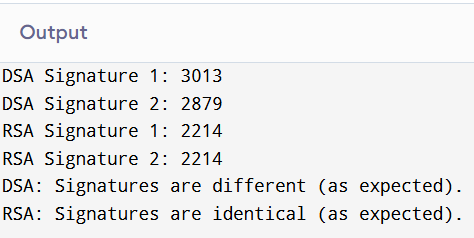
printf("DSA: Signatures are different (as expected).\n");

if (rsa1 == rsa2)

printf("RSA: Signatures are identical (as expected).\n");

return 0;

}

****

**33.**

**CODE:**

#include <stdio.h>

#include <stdint.h>

uint32\_t feistel(uint32\_t half\_block, uint32\_t subkey) {

return (half\_block ^ subkey);

}

void des\_encrypt(uint64\_t plaintext, uint64\_t key, uint64\_t \*ciphertext) {

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

uint32\_t R = plaintext & 0xFFFFFFFF;

uint32\_t round\_keys[16];

for (int i = 0; i < 16; i++) round\_keys[i] = (key >> (i % 28)) & 0xFFFFFFFF;

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

uint32\_t temp = R;

R = L ^ feistel(R, round\_keys[i]);

L = temp;

}

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

}

void des\_decrypt(uint64\_t ciphertext, uint64\_t key, uint64\_t \*plaintext) {

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

uint32\_t R = ciphertext & 0xFFFFFFFF;

uint32\_t round\_keys[16];

for (int i = 0; i < 16; i++) round\_keys[i] = (key >> (i % 28)) & 0xFFFFFFFF;

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

uint32\_t temp = L;

L = R ^ feistel(L, round\_keys[i]);

R = temp;

}

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

}

int main() {

uint64\_t plaintext = 0x0123456789ABCDEF;

uint64\_t key = 0x133457799BBCDFF1;

uint64\_t ciphertext, decrypted;

des\_encrypt(plaintext, key, &ciphertext);

printf("Encrypted: %016llX\n", ciphertext);

des\_decrypt(ciphertext, key, &decrypted);

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

return 0;

}

A screenshot of a computer code

AI-generated content may be incorrect.

**34.**

**CODE:**

#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 8

void pad(char \*data) {

int len = strlen(data);

data[len] = 1;

for (int i = len + 1; i < len + BLOCK\_SIZE; i++) data[i] = 0;

data[len + BLOCK\_SIZE] = '\0';

}

void ecb\_encrypt(const char \*plaintext, char \*ciphertext, const char \*key) {

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

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

ciphertext[i + j] = plaintext[i + j] ^ key[j];

}

}

}

int main() {

char plaintext[64] = "blockdata";

char key[BLOCK\_SIZE] = "12345678";

char ciphertext[64] = {0};

pad(plaintext);

ecb\_encrypt(plaintext, ciphertext, key);

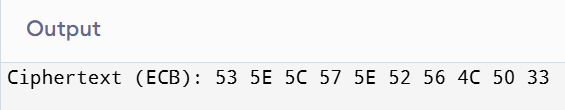
printf("Ciphertext (ECB): ");

for (int i = 0; i < strlen(plaintext); i++) printf("%02X ", (unsigned char)ciphertext[i]);

printf("\n");

return 0;

}

****

**35.**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX 100

void encrypt(char \*plain, int \*key, char \*cipher) {

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

cipher[i] = ((plain[i] - 'A') + key[i]) % 26 + 'A';

}

}

void decrypt(char \*cipher, int \*key, char \*plain) {

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

plain[i] = ((cipher[i] - 'A') - key[i] + 26) % 26 + 'A';

}

}

int main() {

char plaintext[MAX] = "HELLOWORLD";

char ciphertext[MAX] = {0};

char decrypted[MAX] = {0};

int key[MAX];

srand(time(NULL));

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

key[i] = rand() % 26;

}

encrypt(plaintext, key, ciphertext);

decrypt(ciphertext, key, decrypted);

printf("Plaintext: %s\n", plaintext);

printf("Ciphertext: %s\n", ciphertext);

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

return 0;

}

A screenshot of a computer program

AI-generated content may be incorrect.

**36.**

**CODE:**

#include <stdio.h>

#include <string.h>

int gcd(int a, int b) {

while(b) { int t = b; b = a % b; a = t; }

return a;

}

int modInv(int a, int m) {

for(int x = 1; x < m; x++) if((a \* x) % m == 1) return x;

return -1;

}

void affine(char \*in, char \*out, int a, int b, int mode) {

int inv = (mode == 1) ? modInv(a, 26) : a;

for(int i = 0; in[i]; i++) {

char c = in[i];

if(c >= 'A' && c <= 'Z') {

int x = c - 'A';

out[i] = mode ? (inv \* (x - b + 26)) % 26 + 'A' : (a \* x + b) % 26 + 'A';

} else if(c >= 'a' && c <= 'z') {

int x = c - 'a';

out[i] = mode ? (inv \* (x - b + 26)) % 26 + 'a' : (a \* x + b) % 26 + 'a';

} else out[i] = c;

}

out[strlen(in)] = 0;

}

int main() {

int a = 5, b = 8;

if(gcd(a, 26) != 1) {

printf("'a' not coprime with 26\n");

return 1;

}

char text[100], enc[100], dec[100];

printf("Enter text: ");

fgets(text, 100, stdin);

text[strcspn(text, "\n")] = 0; // Remove newline

affine(text, enc, a, b, 0);

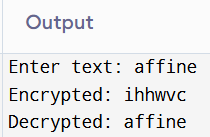
printf("Encrypted: %s\n", enc);

affine(enc, dec, a, b, 1);

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

return 0;

}



**37**

**CODE:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

const char eng\_freq\_order[] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";

void letterFrequency(char \*text, int freq[26]) {

for (int i = 0; i < 26; i++) freq[i] = 0;

for (int i = 0; text[i]; i++) {

if (isalpha(text[i])) {

freq[toupper(text[i]) - 'A']++;

}

}

}

void sortLettersByFreq(int freq[26], char letters[26]) {

for (int i = 0; i < 26; i++) letters[i] = 'A' + i;

for (int i = 0; i < 25; i++) {

for (int j = 0; j < 25 - i; j++) {

if (freq[j] < freq[j + 1]) {

int tmpf = freq[j];

freq[j] = freq[j + 1];

freq[j + 1] = tmpf;

char tmpc = letters[j];

letters[j] = letters[j + 1];

letters[j + 1] = tmpc;

}

}

}

}

void substitute(char \*ciphertext, char \*plaintext, char map[26]) {

int len = strlen(ciphertext);

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

char c = ciphertext[i];

if (isalpha(c)) {

int idx = toupper(c) - 'A';

char sub = map[idx];

plaintext[i] = isupper(c) ? sub : tolower(sub);

} else {

plaintext[i] = c;

}

}

plaintext[len] = '\0';

}

int main() {

char ciphertext[512], plaintext[512];

int freq[26];

char sorted\_letters[26], map[26];

printf("Enter ciphertext:\n");

fgets(ciphertext, sizeof(ciphertext), stdin);

ciphertext[strcspn(ciphertext, "\n")] = 0;

letterFrequency(ciphertext, freq);

sortLettersByFreq(freq, sorted\_letters);

for (int i = 0; i < 26; i++) {

map[sorted\_letters[i] - 'A'] = eng\_freq\_order[i];

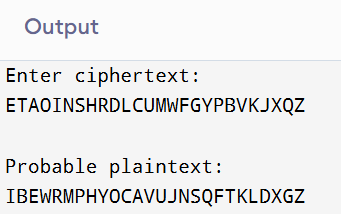
}

substitute(ciphertext, plaintext, map);

printf("\nProbable plaintext:\n%s\n", plaintext);

return 0;

}



**38**

**CODE:**

#include <stdio.h>

int mod26(int x) {

x %= 26;

if (x < 0) x += 26;

return x;

}

int det(int matrix[2][2]) {

return mod26(matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]);

}

int modInverse(int a) {

a = a % 26;

for (int x = 1; x < 26; x++) {

if ((a \* x) % 26 == 1)

return x;

}

return -1;

}

int inverseMatrix(int matrix[2][2], int inv[2][2]) {

int determinant = det(matrix);

int invDet = modInverse(determinant);

if (invDet == -1) return 0; // Not invertible

inv[0][0] = mod26(matrix[1][1] \* invDet);

inv[0][1] = mod26(-matrix[0][1] \* invDet);

inv[1][0] = mod26(-matrix[1][0] \* invDet);

inv[1][1] = mod26(matrix[0][0] \* invDet);

return 1;

}

void multiply(int A[2][2], int B[2][2], int res[2][2]) {

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

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

res[i][j] = 0;

for (int k=0; k<2; k++)

res[i][j] += A[i][k]\*B[k][j];

res[i][j] = mod26(res[i][j]);

}

}

void printMatrix(int m[2][2]) {

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

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

printf("%3d ", m[i][j]);

printf("\n");

}

}

int main() {

int P[2][2] = {{7, 4}, {19, 19}};

int C[2][2] = {{2, 19}, {5, 7}};

int P\_inv[2][2], K[2][2];

if (!inverseMatrix(P, P\_inv)) {

printf("Plaintext matrix is not invertible mod 26\n");

return 1;

}

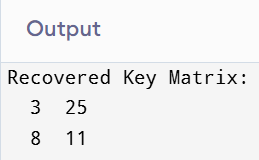
multiply(C, P\_inv, K);

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

printMatrix(K);

return 0;

}



**39**

**CODE:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

const double eng\_freq[26] = {

8.167,1.492,2.782,4.253,12.702,2.228,2.015,6.094,

6.966,0.153,0.772,4.025,2.406,6.749,7.507,1.929,

0.095,5.987,6.327,9.056,2.758,0.978,2.360,0.150,

1.974,0.074

};

double score(char \*text) {

int freq[26] = {0}, len = 0;

for (int i = 0; text[i]; i++)

if (isalpha(text[i])) freq[toupper(text[i])-'A']++, len++;

if (len == 0) return 0;

double s = 0;

for (int i = 0; i < 26; i++)

s += (freq[i] \* 100.0 / len) \* eng\_freq[i];

return s;

}

void decrypt(char \*ct, char \*pt, int key) {

for (int i = 0; ct[i]; i++) {

char c = ct[i];

if (isupper(c)) pt[i] = ((c - 'A' - key + 26) % 26) + 'A';

else if (islower(c)) pt[i] = ((c - 'a' - key + 26) % 26) + 'a';

else pt[i] = c;

}

pt[strlen(ct)] = 0;

}

int main() {

char ct[512] = "Wklv lv d whvw phvvdjh."; // <--- input is hardcoded

char pt[512];

double best\_score = -1;

int best\_key = 0;

for (int k = 0; k < 26; k++) {

decrypt(ct, pt, k);

double sc = score(pt);

if (sc > best\_score) {

best\_score = sc;

best\_key = k;

}

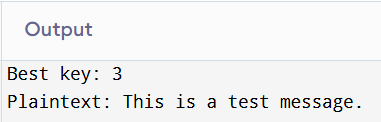
}

decrypt(ct, pt, best\_key);

printf("Best key: %d\nPlaintext: %s\n", best\_key, pt);

return 0;

}



**40.**

**Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#define MAX 512

#define TOP 10

const char eng\_freq[] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";

typedef struct {

char text[MAX];

double score;

} Candidate;

void freq(const char \*txt, double f[26]) {

int c[26] = {0}, tot = 0;

for (int i=0; txt[i]; i++)

if (isalpha(txt[i])) c[toupper(txt[i])-'A']++, tot++;

for(int i=0; i<26; i++) f[i] = tot ? (double)c[i]/tot : 0;

}

double score\_text(const char \*txt) {

double f[26], s=0;

freq(txt,f);

for(int i=0; i<26; i++) s += f[eng\_freq[i]-'A'] \* (26 - i);

return s;

}

void substitute(const char \*c, char \*p, char map[26]) {

for(int i=0; c[i]; i++) {

if (isalpha(c[i])) {

char ch = map[toupper(c[i])-'A'];

p[i] = islower(c[i]) ? tolower(ch) : ch;

} else p[i] = c[i];

}

p[strlen(c)] = 0;

}

int cmp(const void \*a, const void \*b) {

return ((Candidate\*)b)->score - ((Candidate\*)a)->score > 0 ? 1 : -1;

}

int main() {

char ct[MAX];

Candidate cands[TOP];

char map[26];

printf("Enter ciphertext:\n");

fgets(ct, MAX, stdin);

ct[strcspn(ct,"\n")] = 0;

for(int shift=0; shift<TOP; shift++) {

for(int i=0; i<26; i++) map[i] = eng\_freq[(i+shift)%26];

substitute(ct, cands[shift].text, map);

cands[shift].score = score\_text(cands[shift].text);

}

qsort(cands, TOP, sizeof(Candidate), cmp);

printf("\nTop %d plaintext guesses:\n", TOP);

for(int i=0; i<TOP; i++)

printf("[%d] Score: %.3f\n%s\n\n", i+1, cands[i].score, cands[i].text);

return 0;

}

