1. **Write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25**

#include <stdio.h>

int main()

{

char text[100];

char encrypted[100];

char decrypted[100];

int key, i;

printf("Enter text : ");

scanf("%s", text);

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

scanf("%d", &key);

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

{

char ch = text[i];

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

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

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

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

else if (ch == '\_')

encrypted[i] = ' ';

else

encrypted[i] = ch;

}

encrypted[i] = '\0';

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

{

char ch = encrypted[i];

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

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

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

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

else if (ch == ' ')

decrypted[i] = '\_';

else

decrypted[i] = ch;

}

decrypted[i] = '\0';

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

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

return 0;

}

1. **Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.**

#include <stdio.h>

int main() {

char text[100];

char encrypted[100];

char decrypted[100];

int a, b, a\_inv = 0, i;

printf("Enter text : ");

scanf("%s", text);

printf("Enter key 'a' (must be coprime with 26): ");

scanf("%d", &a);

printf("Enter key 'b': ");

scanf("%d", &b);

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

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

a\_inv = i;

break;

}

}

if (a\_inv == 0) {

printf("Invalid key 'a'. No modular inverse exists.\n");

return 1;

}

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

char ch = text[i];

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

encrypted[i] = ((a \* (ch - 'A') + b) % 26) + 'A';

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

encrypted[i] = ((a \* (ch - 'a') + b) % 26) + 'a';

else if (ch == '\_')

encrypted[i] = ' ';

else

encrypted[i] = ch;

}

encrypted[i] = '\0';

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

{

char ch = encrypted[i];

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

decrypted[i] = (a\_inv \* ((ch - 'A' - b + 26) % 26)) % 26 + 'A';

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

decrypted[i] = (a\_inv \* ((ch - 'a' - b + 26) % 26)) % 26 + 'a';

else if (ch == ' ')

decrypted[i] = ' ';

else

decrypted[i] = ch;

}

decrypted[i] = '\0';

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

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

return 0;

}

1. **Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.**

#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;

}

1. **Write a C program for polyalphabetic substitution cipher uses a separate monoalphabetic substitution cipher for each successive letter of plaintext, depending on a key.**

#include <stdio.h>

#include <string.h>

int main()

{

char text[100], key[100], encrypted[100], decrypted[100];

int i, j = 0, shift;

printf("Enter text : ");

scanf("%s", text);

printf("Enter key (letters only): ");

scanf("%s", key);

int len = strlen(text);

int keyLen = strlen(key);

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

{

if (text[i] == '\_')

{

encrypted[i] = ' ';

}

else

{

shift = (key[j % keyLen] - 'a') % 26;

encrypted[i] = ((text[i] - 'a' + shift) % 26) + 'a';

j++;

}

}

encrypted[i] = '\0';

j = 0;

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

{

if (encrypted[i] == ' ')

{

decrypted[i] = '\_';

}

else

{

shift = (key[j % keyLen] - 'a') % 26;

decrypted[i] = ((encrypted[i] - 'a' - shift + 26) % 26) + 'a';

j++;

}

}

decrypted[i] = '\0';

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

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

return 0;

}

**5. Write a C program for generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter C: C = E([a, b], p) = (ap + b) mod 26 A basic requirement of any encryption algorithm is that it be one-to-one. That is, if p q, then E(k, p) E(k, q). Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to-one for all values of a. For example, for a = 2 and b = 3, then E([a, b], 0) = E([a, b], 13) = 3.**

**a. Are there any limitations on the value of b?**

**b. Determine which values of a are not allowed.**

#include <stdio.h>

int main()

{

char plaintext[100], ciphertext[100], decrypted[100];

int a, b, i, a\_inv = -1;

printf("Enter the value of a : ");

scanf("%d", &a);

int temp\_a = a, temp\_b = 26;

while (temp\_b != 0)

{

int temp = temp\_b;

temp\_b = temp\_a % temp\_b;

temp\_a = temp;

}

if (temp\_a != 1)

{

printf("Error: 'a' must be co-prime with 26.\n");

return 1;

}

printf("Enter the value of b: ");

scanf("%d", &b);

printf("Enter the plaintext: ");

scanf("%s", plaintext);

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

{

if (plaintext[i] >= 'a' && plaintext[i] <= 'z')

{

ciphertext[i] = (a \* (plaintext[i] - 'a') + b) % 26 + 'a';

}

else

{

printf("Invalid character in plaintext. Only lowercase letters are allowed.\n");

return 1;

}

}

ciphertext[i] = '\0';

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

for (int j = 1; j < 26; j++)

{

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

{

a\_inv = j;

break;

}

}

if (a\_inv == -1) {

printf("Modular inverse of a does not exist. Decryption not possible.\n");

return 1;

}

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

decrypted[i] = (a\_inv \* ((ciphertext[i] - 'a' - b + 26)) % 26) + 'a';

}

decrypted[i] = '\0';

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

return 0;

}

1. **Write a C program for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.**

#include <stdio.h>

int main()

{

char ct[100];

int y1 = 1, y2 = 20, x1 = 4, x2 = 19;

int a, b, a\_inv, x, y;

printf("Enter ciphertext : ");

scanf("%s", ct);

for (a = 1; a < 26; a++)

{

if ((a \* ((x1 - x2 + 26) % 26)) % 26 == (y1 - y2 + 26) % 26)

break;

}

for (a\_inv = 1; a\_inv < 26; a\_inv++)

{

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

break;

}

b = (y1 - a \* x1 + 26 \* 26) % 26;

printf("Decrypted text: ");

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

{

if (ct[i] >= 'A' && ct[i] <= 'Z')

{

y = ct[i] - 'A';

x = (a\_inv \* (y - b + 26)) % 26;

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

}

else

{

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

}

}

printf("\n");

return 0;

**7. Write a C program for the following ciphertext was generated using a simple substitution algorithm.**

**53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡\*8†83 (88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2(5\*—4)8¶8\***

**;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 (‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;**

**Decrypt this message.**

**1. As you know, the most frequently occurring letter in English is e. Therefore, the first or second (or perhaps third?) most common character in the message is likely to stand for e. Also, e is often seen in pairs (e.g., meet, fleet, speed, seen, been,**

**agree, etc.). Try to find a character in the ciphertext that decodes to e.**

**2. The most common word in English is “the.” Use this fact to guess the characters that stand for t and h.**

**3. Decipher the rest of the message by deducing additional words.**

#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;

}

**8. Write a C program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated.**

**For example, using the keyword *CIPHER*, write out the keyword followed by unused letters in normal order and match this against the plaintext letters:**

#include <stdio.h>

#include <string.h>

int main()

{

char plaintext[100], ciphertext[100], keyword[100];

char alphabet[] ="abcdefghijklmnopqrstuvwxyz";

char cipher[26];

int i, j, k = 0;

printf("Enter the keyword: ");

scanf("%s", keyword);

printf("Enter the plaintext: ");

scanf("%s", plaintext);

for(i=0;i<strlen(keyword);i++)

{

if(strchr(cipher,keyword[i])==NULL)

{

cipher[k++]=keyword[i];

}

}

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

{

if(strchr(keyword,alphabet[i])==NULL)

{

cipher[k++]=alphabet[i];

}

}

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

{

ciphertext[i]=cipher[plaintext[i]-'a'];

}

ciphertext[i]='\0';

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

return 0;

}

**9.Write a C program for PT-109 American patrol boat, under the command of Lieutenant John F. Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in Playfair code:**

**KXJEY UREBE ZWEHE WRYTU HEYFS**

**KREHE GOYFI WTTTU OLKSY CAJPO**

**BOTEI ZONTX BYBNT GONEY CUZWR**

**GDSON SXBOU YWRHE BAAHY USEDQ**

#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;

}

**10. Write a C program for Playfair matrix:**

**M F H I/J K**

**U N O P Q**

**Z V W X Y**

**E L A R G**

**D S T B C**

**Encrypt this message: Must see you over Cadogan West. Coming at once.**

#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;

}

**11. Write a C program for possible keys does the Playfair cipher have? Ignore the fact that some keys might produce identical encryption results. Express your answer as an approximate power of 2.**

**a. Now take into account the fact that some Playfair keys produce the same encryption results. How many effectively unique keys does the Playfair cipher have?**

#include <stdio.h>

#include <math.h>

int main()

{

int i;

double log2\_factorial = 0.0;

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

{

log2\_factorial += log2(i);

}

printf("The number of possible Playfair cipher keys is approximately 2^%.2f\n", log2\_factorial);

return 0;

}

**12. a. Write a C program to Encrypt the message “meet me at the usual place at ten rather than eight oclock” using the Hill cipher with the key.**

**9 4**

**5 7**

**a. Show your calculations and the result.**

**b. Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext.**

#include <stdio.h>

#include <string.h>

int main() {

char ciphertext[1000];

int freq[26] = {0};

char english\_freq[] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";

int i, j, len, top\_n;

printf("Enter ciphertext : ");

scanf("%s", ciphertext);

printf("How many top guesses to print? ");

scanf("%d", &top\_n);

len = strlen(ciphertext);

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

if(ciphertext[i] >= 'A' && ciphertext[i] <= 'Z')

freq[ciphertext[i] - 'A']++;

}

int sorted[26];

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

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

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

if(freq[sorted[j]] > freq[sorted[i]]) {

int temp = sorted[i];

sorted[i] = sorted[j];

sorted[j] = temp;

}

}

}

printf("\nTop %d possible plaintexts:\n", top\_n);

for(i = 0; i < top\_n; i++) {

int key = (sorted[0] - (english\_freq[i] - 'A') + 26) % 26;

printf("Guess %d (key = %d): ", i+1, key);

for(j = 0; j < len; j++) {

if(ciphertext[j] >= 'A' && ciphertext[j] <= 'Z') {

char p = (ciphertext[j] - 'A' - key + 26) % 26 + 'A';

printf("%c", p);

} else {

printf("%c", ciphertext[j]);

}

}

printf("\n");

}

return 0;

}

**13. Write a C 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 <string.h>

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 main()

{

char input[] = "meet me";

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

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

int invKey[2][2];

int i, j = 0, len;

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

if (input[i] >= 'a' && input[i] <= 'z') {

plain[j++] = input[i];

}

}

if (j % 2 != 0) {

plain[j++] = 'x';

}

plain[j] = '\0';

len = j;

int det = key[0][0] \* key[1][1] - key[0][1] \* key[1][0];

det = (det % 26 + 26) % 26;

int detInv = modInverse(det, 26);

if (detInv == -1) {

printf("Key matrix is not invertible under mod 26.\n");

return 1;

}

invKey[0][0] = key[1][1];

invKey[0][1] = -key[0][1];

invKey[1][0] = -key[1][0];

invKey[1][1] = key[0][0];

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

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

invKey[i][j] = (invKey[i][j] \* detInv) % 26;

if (invKey[i][j] < 0)

invKey[i][j] += 26;

}

}

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

int a = plain[i] - 'a';

int b = plain[i + 1] - 'a';

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

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

}

cipher[len] = '\0';

printf("Cipher Text: %s\n", cipher);

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

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

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

int x = invKey[0][0] \* a + invKey[0][1] \* b;

int y = invKey[1][0] \* a + invKey[1][1] \* b;

while (x < 0) x += 26;

while (y < 0) y += 26;

decrypted[i] = (x % 26) + 'a';

decrypted[i + 1] = (y % 26) + 'a';

}

decrypted[len] = '\0';

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

return 0;

}

**14. Write a C program for one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 0 and 26. For example, if the key is 3 19 5 . . . , then the first letter of plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.**

**a. Encrypt the plaintext send more money with the key stream**

**9 0 1 7 23 15 21 14 11 11 2 8 9**

**b. Using the ciphertext produced in part (a), find a key so that the cipher text decrypts to the plaintext cash not needed.**

#include <stdio.h>

int main() {

char plaintext[] = "sendmoremoney";

int key[] = {9,0,1,7,23,15,21,14,11,11,2,8,9};

char ciphertext[100], decrypted[100];

int i;

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

if (plaintext[i] >= 'A' && plaintext[i] <= 'Z') {

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

}

else if (plaintext[i] >= 'a' && plaintext[i] <= 'z') {

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

}

else {

ciphertext[i] = plaintext[i];

}

}

ciphertext[i] = '\0';

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

if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {

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

}

else if (ciphertext[i] >= 'a' && ciphertext[i] <= 'z') {

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

}

else {

decrypted[i] = ciphertext[i];

}

}

decrypted[i] = '\0';

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

printf("Key : ");

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

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

}

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

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

return 0;

}

**15. Write a C 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 <string.h>

int main() {

char ciphertext[1000];

int freq[26] = {0};

char english\_freq[] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";

int i, j, len, top\_n;

printf("Enter ciphertext : ");

scanf("%s", ciphertext);

printf("How many top guesses to print? ");

scanf("%d", &top\_n);

len = strlen(ciphertext);

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

if(ciphertext[i] >= 'A' && ciphertext[i] <= 'Z')

freq[ciphertext[i] - 'A']++;

}

int sorted[26];

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

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

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

if(freq[sorted[j]] > freq[sorted[i]]) {

int temp = sorted[i];

sorted[i] = sorted[j];

sorted[j] = temp;

}

}

}

printf("\nTop %d possible plaintexts:\n", top\_n);

for(i = 0; i < top\_n; i++) {

int key = (sorted[0] - (english\_freq[i] - 'A') + 26) % 26;

printf("Guess %d (key = %d): ", i+1, key);

for(j = 0; j < len; j++) {

if(ciphertext[j] >= 'A' && ciphertext[j] <= 'Z') {

char p = (ciphertext[j] - 'A' - key + 26) % 26 + 'A';

printf("%c", p);

} else {

printf("%c", ciphertext[j]);

}

}

printf("\n");

}

return 0;

}

**16. Write a C program that can perform a letter frequency attack on any monoalphabetic substitution 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 <string.h>

int main() {

char ciphertext[10000];

char ciphertext\_copy[10000];

int freq[26];

char cipher\_freq[26];

char english\_freq[] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";

char mapping[26];

int i, j, k, n;

char temp;

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

freq[i] = 0;

cipher\_freq[i] = 'A' + i;

}

printf("Enter ciphertext (only letters will be considered):\n");

fgets(ciphertext, 10000, stdin);

printf("Enter number of top likely plaintexts to display: ");

scanf("%d", &n);

if (n > 10) n = 10;

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

char ch = ciphertext[i];

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

if (ch >= 'a' && ch <= 'z') ch = ch - ('a' - 'A');

freq[ch - 'A']++;

}

}

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

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

if (freq[j] > freq[i]) {

int ftemp = freq[i];

freq[i] = freq[j];

freq[j] = ftemp;

temp = cipher\_freq[i];

cipher\_freq[i] = cipher\_freq[j];

cipher\_freq[j] = temp;

}

}

}

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

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

mapping[i] = '?';

}

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

int shift = (i + k) % 26;

mapping[cipher\_freq[i] - 'A'] = english\_freq[shift];

}

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

char ch = ciphertext[i];

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

int lower = 0;

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

ch = ch - ('a' - 'A');

lower = 1;

}

char mapped = mapping[ch - 'A'];

if (lower) ciphertext\_copy[i] = mapped + ('a' - 'A');

else ciphertext\_copy[i] = mapped;

} else {

ciphertext\_copy[i] = ch;

}

}

ciphertext\_copy[i] = '\0';

printf("\nPossible plaintext %d:\n%s\n", k + 1, ciphertext\_copy);

}

return 0;

}

**17. Write a C 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>

#define DES\_BLOCK\_SIZE 8

#define NUM\_KEYS 16

void generateKeys(uint64\_t key, uint64\_t keys[NUM\_KEYS]) {

}

void desDecrypt(uint64\_t ciphertext, uint64\_t keys[NUM\_KEYS], uint64\_t \*plaintext) {

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

}

\*plaintext = ciphertext;

}

int main() {

uint64\_t key = 0x133457799BBCDFF1;

uint64\_t ciphertext = 0x0123456789ABCDEF;

uint64\_t keys[NUM\_KEYS];

uint64\_t plaintext;

generateKeys(key, keys);

desDecrypt(ciphertext, keys, &plaintext);

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

return 0;

}

**18. Write a C 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>

void generateSubkeys(uint64\_t key)

{

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

uint32\_t right = key & 0xFFFFFFF;

uint64\_t subkey;

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

{

left = (left << 1) | (left >> 27);

right = (right << 1) | (right >> 27);

subkey = ((left & 0xFFFFFFF) << 24) | (right & 0xFFFFFFF);

printf("Subkey %d: %016llX\n", i + 1, subkey);

}

}

int main()

{

uint64\_t key = 0x0123456789ABCDEF;

generateSubkeys(key);

return 0;

}

**19. Write a C 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 <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;

}

**20. Write a C program for ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C1 obviously corrupts P1 and P2.**

**a. Are any blocks beyond P2 affected?**

**b. Suppose that there is a bit error in the source version of P1. Through how manyciphertext blocks is this error propagated? What is the effect at the receiver?**

#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;

}

**21. Write a C 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 <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;

}

**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 <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;

}

**23. Write a C program for Encrypt and decrypt in counter mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES. Test data for S-DES using a counter starting at 0000 0000. A binary plaintext of 0000 0001 0000 0010 0000 0100 encrypted with a binary key of 01111 11101 should give a binary plaintext of 0011 1000 0100 1111 0011 0010. Decryption should work**

**correspondingly.**

#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;

}

**24. Write a C 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 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;

}

**25. Write a C program for set of blocks encoded with the RSA algorithm and we don’t have the private key. Assume n = pq, e is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n. Does this help us in any way?**

#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;

}