**CRYPTOGRAPY LAB PROGRAMS**

**1.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.**

**Code;**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MOD 26

void text\_to\_numbers(char \*text, int \*numbers, int n) {

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

numbers[i] = text[i] - 'A';

}

void mod\_inverse(int P[4][4], int P\_inv[4][4]) {

int det = (P[0][0] \* P[1][1] - P[0][1] \* P[1][0]) % MOD;

if (det < 0) det += MOD;

int det\_inv = -1;

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

if ((det \* i) % MOD == 1) {

det\_inv = i;

break;

}

P\_inv[0][0] = P[1][1] \* det\_inv % MOD;

P\_inv[0][1] = -P[0][1] \* det\_inv % MOD;

P\_inv[1][0] = -P[1][0] \* det\_inv % MOD;

P\_inv[1][1] = P[0][0] \* det\_inv % MOD;

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

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

if (P\_inv[i][j] < 0) P\_inv[i][j] += MOD;

}

void recover\_key(char plain\_texts[2][3], char cipher\_texts[2][3], int key[2][2]) {

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

text\_to\_numbers(plain\_texts[0], P[0], 2);

text\_to\_numbers(plain\_texts[1], P[1], 2);

text\_to\_numbers(cipher\_texts[0], C[0], 2);

text\_to\_numbers(cipher\_texts[1], C[1], 2);

mod\_inverse(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() {

char plain\_texts[2][3] = {"HE", "CO"};

char cipher\_texts[2][3] = {"JP", "LQ"};

int key[2][2];

recover\_key(plain\_texts, cipher\_texts, key);

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

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

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

printf("\n");

}

return 0;

}

**2.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.**

**Code;**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define ALPHABET\_SIZE 26

#define MAX\_PLAINTEXTS 10

const float english\_freq[ALPHABET\_SIZE] = {

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

};

typedef struct {

char text[1000];

float score;

} Decryption;

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

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

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

plain[i] = ((cipher[i] - 'A' - key + ALPHABET\_SIZE) % ALPHABET\_SIZE) + 'A';

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

plain[i] = ((cipher[i] - 'a' - key + ALPHABET\_SIZE) % ALPHABET\_SIZE) + 'a';

else

plain[i] = cipher[i];

}

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

}

float score\_text(const char \*text) {

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

int total = 0;

float score = 0.0;

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

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

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

total++;

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

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

total++;

}

}

if (total == 0) return 10000.0;

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

float observed = (float)freq[i] / total \* 100;

score += (observed - english\_freq[i]) \* (observed - english\_freq[i]);

}

return score;

}

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

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

}

void frequency\_attack(const char \*cipher, int top\_n) {

Decryption results[ALPHABET\_SIZE];

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

decrypt(cipher, key, results[key].text);

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

}

qsort(results, ALPHABET\_SIZE, sizeof(Decryption), compare);

for (int i = 0; i < top\_n && i < ALPHABET\_SIZE; i++)

printf("%d: %s\n", i + 1, results[i].text);

}

int main() {

char cipher[1000];

int top\_n;

printf("Enter the encrypted text: ");

fgets(cipher, sizeof(cipher), stdin);

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

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

if (scanf("%d", &top\_n) != 1 || top\_n <= 0 || top\_n > ALPHABET\_SIZE) {

printf("Invalid input. Please enter a valid number between 1 and %d.\n", ALPHABET\_SIZE);

return 1;

}

frequency\_attack(cipher, top\_n);

return 0;

**3.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.**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdint.h>

#define ROUNDS 16

// Initial Permutation Table

static const int IP[] = {

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

};

// Final Permutation Table

static const int FP[] = {

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

};

// Key schedule shifts

static const int key\_shifts[] = {

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

};

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

uint64\_t permuted = 0;

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

permuted |= ((\*block >> (64 - table[i])) & 1) << (n - 1 - i);

}

\*block = permuted;

}

void generate\_decryption\_keys(uint64\_t key, uint64\_t keys[ROUNDS]) {

uint64\_t permuted\_key = 0;

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

permuted\_key |= ((key >> (64 - i - 1)) & 1) << (55 - i);

}

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

uint32\_t D = permuted\_key & 0xFFFFFFF;

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

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

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

keys[ROUNDS - 1 - i] = ((uint64\_t)C << 28) | D;

}

}

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

uint64\_t keys[ROUNDS];

generate\_decryption\_keys(key, keys);

permute(&ciphertext, IP, 64);

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

uint32\_t R = ciphertext & 0xFFFFFFFF;

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

uint32\_t temp = R;

R = L ^ (R + keys[i]);

L = temp;

}

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

permute(&pre\_output, FP, 64);

return pre\_output;

}

int main() {

uint64\_t ciphertext = 0xAABB09182736CCDD;

uint64\_t key = 0x133457799BBCDFF1;

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

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

return 0;

}

**4.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.**

**code:**

#include <stdio.h>

#include <stdint.h>

int initial\_permutation[64] = { ... };

int final\_permutation[64] = { ... };

int expansion\_table[48] = { ... };

int s\_boxes[8][4][16] = { ... };

int permutation\_table[32] = { ... };

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

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

uint32\_t left = (key >> 36) & 0xFFFFFFF;

uint32\_t right = (key >> 8) & 0xFFFFFFF;

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

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

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

subkeys[i] = ((uint64\_t)left << 28) | right;

}

}

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

uint64\_t subkeys[16];

generate\_subkeys(key, subkeys);

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

subkeys[i] = subkeys[15 - i];

}

}

int main() {

uint64\_t ciphertext = 0x85E813540F0AB405;

uint64\_t key = 0x133457799BBCDFF1;

des\_decrypt(ciphertext, key);

return 0;

}

**5. 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?**

**Code:**

#include <stdio.h>

#include <stdint.h>

#include <string.h>

#include <openssl/des.h>

void triple\_des\_encrypt(const uint8\_t \*plaintext, uint8\_t \*ciphertext, const uint8\_t \*key, const uint8\_t \*iv) {

DES\_cblock key1, key2, key3, iv\_copy;

memcpy(key1, key, 8);

memcpy(key2, key + 8, 8);

memcpy(key3, key + 16, 8);

memcpy(iv\_copy, iv, 8);

DES\_key\_schedule ks1, ks2, ks3;

DES\_set\_key\_unchecked(&key1, &ks1);

DES\_set\_key\_unchecked(&key2, &ks2);

DES\_set\_key\_unchecked(&key3, &ks3);

DES\_ede3\_cbc\_encrypt(plaintext, ciphertext, 8, &ks1, &ks2, &ks3, &iv\_copy, DES\_ENCRYPT);

}

int main() {

uint8\_t key[24] = "thisisaverystrongkey!";

uint8\_t iv[8] = "initvect";

uint8\_t plaintext[8] = "message";

uint8\_t ciphertext[8];

triple\_des\_encrypt(plaintext, ciphertext, key, iv);

printf("Ciphertext: ");

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

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

}

printf("\n");

return 0;

}