CRYPTOGRAPY LAB PROGRAMS

MALREDDY-192372015

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;
```

```c
    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;
}
```