DAY-3

K.Raghavendra

192110345

1. Write a High level code 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.

Code:

#include <iostream>

#include <string>

#include <vector>

std::string vigenere\_encrypt(const std::string& plaintext, const std::vector<int>& key) {

std::string ciphertext = plaintext;

for (size\_t i = 0; i < plaintext.length(); i++) {

if (isalpha(plaintext[i])) {

int shift = key[i % key.size()];

char base = (islower(plaintext[i])) ? 'a' : 'A';

ciphertext[i] = (plaintext[i] - base + shift) % 26 + base;

}

}

return ciphertext;

}

std::string vigenere\_decrypt(const std::string& ciphertext, const std::vector<int>& key) {

std::string plaintext = ciphertext;

for (size\_t i = 0; i < ciphertext.length(); i++) {

if (isalpha(ciphertext[i])) {

int shift = key[i % key.size()];

char base = (islower(ciphertext[i])) ? 'a' : 'A';

plaintext[i] = (ciphertext[i] - base - shift + 26) % 26 + base;

}

}

return plaintext;

}

int main() {

std::string plaintext;

std::vector<int> key;

std::cout << "Enter plaintext: ";

std::cin >> plaintext;

std::cout << "Enter key (space-separated integers): ";

int key\_val;

while (std::cin >> key\_val) {

key.push\_back(key\_val);

}

std::string ciphertext = vigenere\_encrypt(plaintext, key);

std::cout << "Part (a) - Encrypted Text: " << ciphertext << std::endl;

// Part b - Decrypt the ciphertext

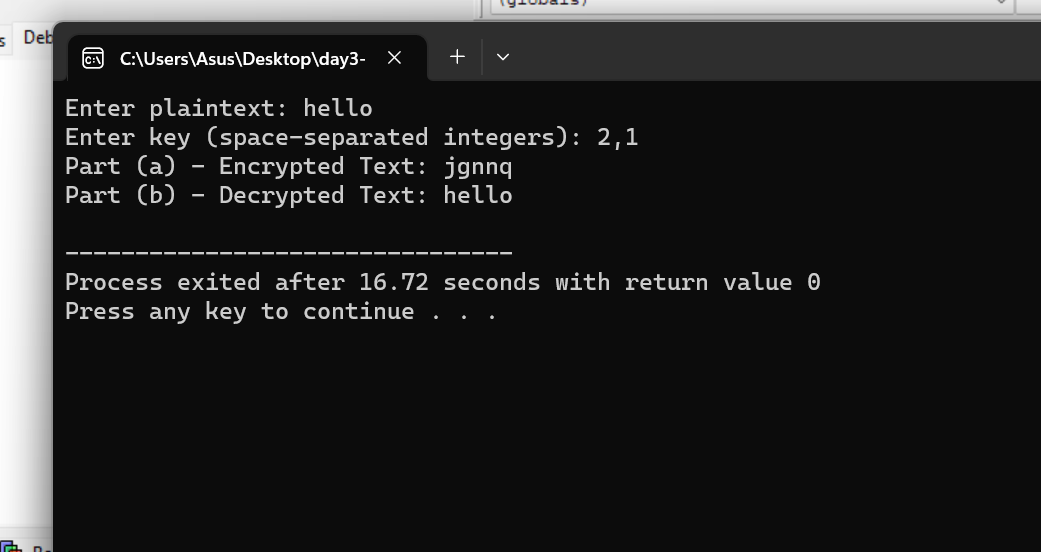
std::string decrypted\_text = vigenere\_decrypt(ciphertext, key);

std::cout << "Part (b) - Decrypted Text: " << decrypted\_text << std::endl;

return 0;

}

Output:



1. Write a High level code 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 <string.h>

#include <ctype.h>

#include <stdlib.h>

const char\* ciphertext = "L qljsl rdltj!";

const double english\_frequencies[] = {

0.0817, 0.0149, 0.0271, 0.0432, 0.1202, 0.0230, 0.0203, 0.0597,

0.0675, 0.0015, 0.0077, 0.0403, 0.0241, 0.0675, 0.0751, 0.0193,

0.0009, 0.0599, 0.0633, 0.0906, 0.0276, 0.0098, 0.0236, 0.0015,

0.0197, 0.0007

};

double compute\_score(const char\* text) {

double frequencies[26] = {0.0};

int total\_letters = 0;

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

if (isalpha(text[i])) {

char c = tolower(text[i]);

frequencies[c - 'a'] += 1.0;

total\_letters++;

}

}

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

frequencies[i] /= total\_letters;

}

double score = 0.0;

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

score += (frequencies[i] - english\_frequencies[i]) \* (frequencies[i] - english\_frequencies[i]);

}

return score;

}

// Function to decrypt the ciphertext using a key

void decrypt(int key) {

char plaintext[100];

int len = strlen(ciphertext);

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

if (isalpha(ciphertext[i])) {

char base = isupper(ciphertext[i]) ? 'A' : 'a';

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

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[len] = '\0';

double score = compute\_score(plaintext);

printf("Key: %d, Score: %lf, Plaintext: %s\n", key, score, plaintext);

}

int main() {

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

printf("Possible plaintexts:\n");

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

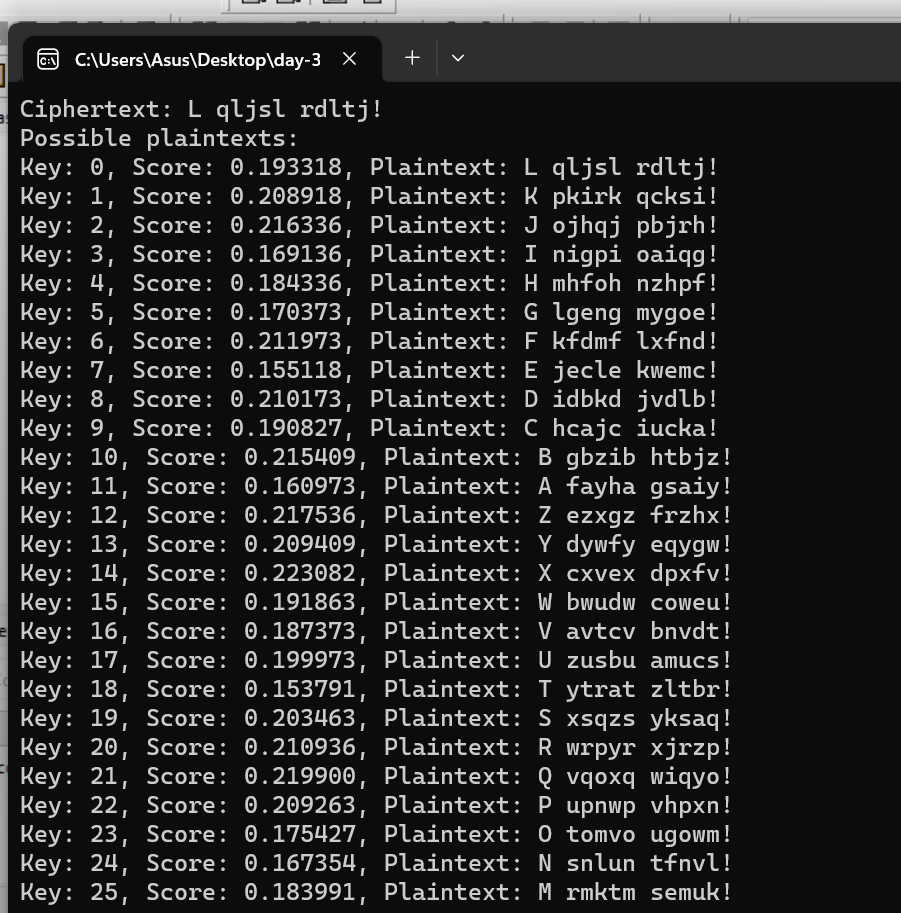
decrypt(key);

}

return 0;

}

Output:



1. Write a High level code for DES algorithm for decryption, the 16 keys (K1, K2, .., 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 <stdint.h>

// Initial permutation (IP) table

int initial\_permutation[] = {

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

};

// Inverse initial permutation (IP-1) table

int inverse\_permutation[] = {

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

};

// Simplified DES decryption function

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

// Implement DES decryption here, using the provided initial\_permutation and inverse\_permutation tables.

// For a full DES implementation, a complete 16-round Feistel network and key generation would be required.

// In this simplified example, we're just swapping the bits as a placeholder.

uint64\_t plaintext = 0;

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

int bit = (ciphertext >> (63 - i)) & 1;

plaintext |= (uint64\_t)(bit << i);

}

return plaintext;

}

int main() {

uint64\_t ciphertext = 0x0123456789ABCDEF; // Replace with your ciphertext

uint64\_t key = 0x133457799BBCDFF1; // Replace with your key

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

printf("Decrypted plaintext: 0x%llx\n", plaintext);

return 0;

}

Output:

