**Day-3 Lab programs**

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.**

**C program:**

#include <stdio.h>

#include <string.h>

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

int i;

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

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

int shift = encrypt ? key\_stream[i] : -key\_stream[i];

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

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

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

int shift = encrypt ? key\_stream[i] : -key\_stream[i];

char c = plaintext[i] - 'A';

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

} else {

ciphertext[i] = plaintext[i];

}

}

ciphertext[i] = '\0';

}

int main() {

char plaintext[] = "send more money";

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

char ciphertext[100];

vigenere\_cipher(plaintext, key\_stream, ciphertext, 1);

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

char decrypted\_text[100];

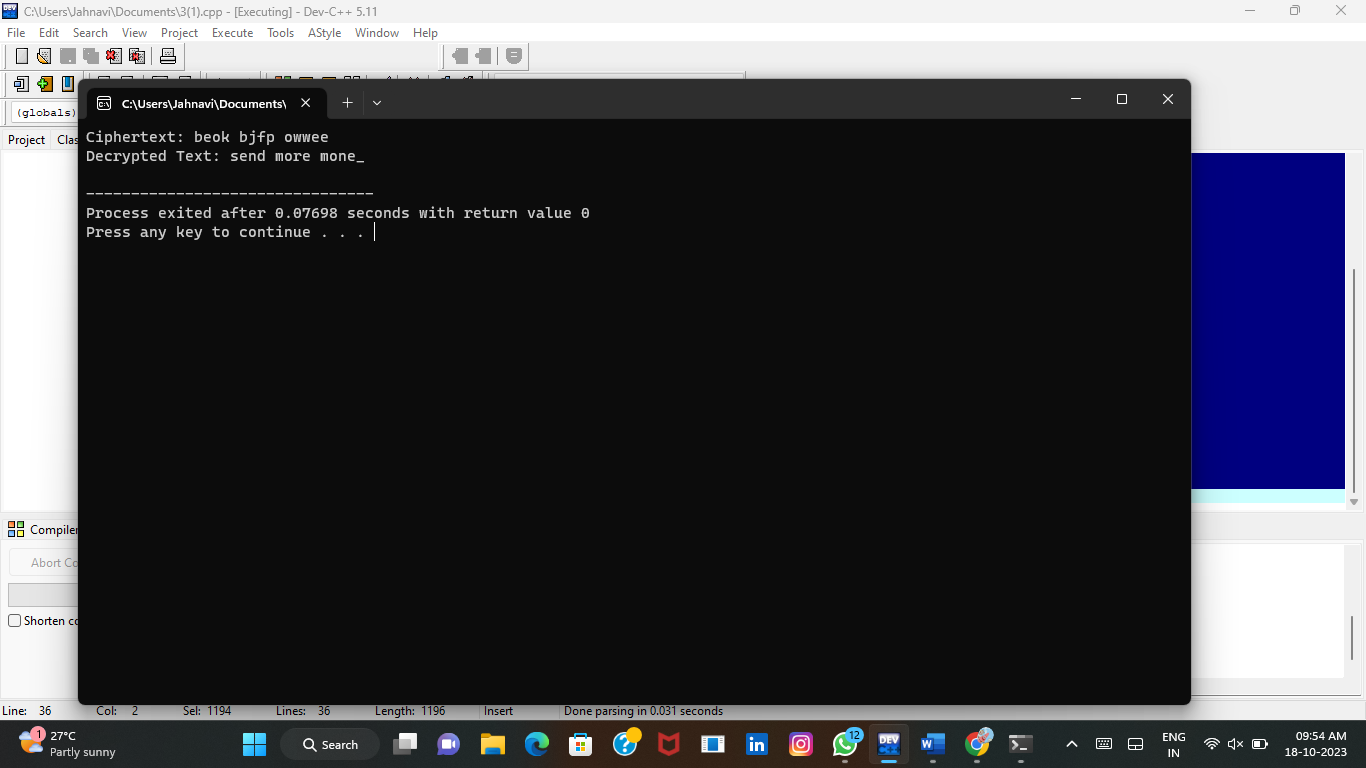
vigenere\_cipher(ciphertext, key\_stream, decrypted\_text, 0);

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

return 0;

}

**Output:**



**2.** **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.”**

**C program:**

#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++) {

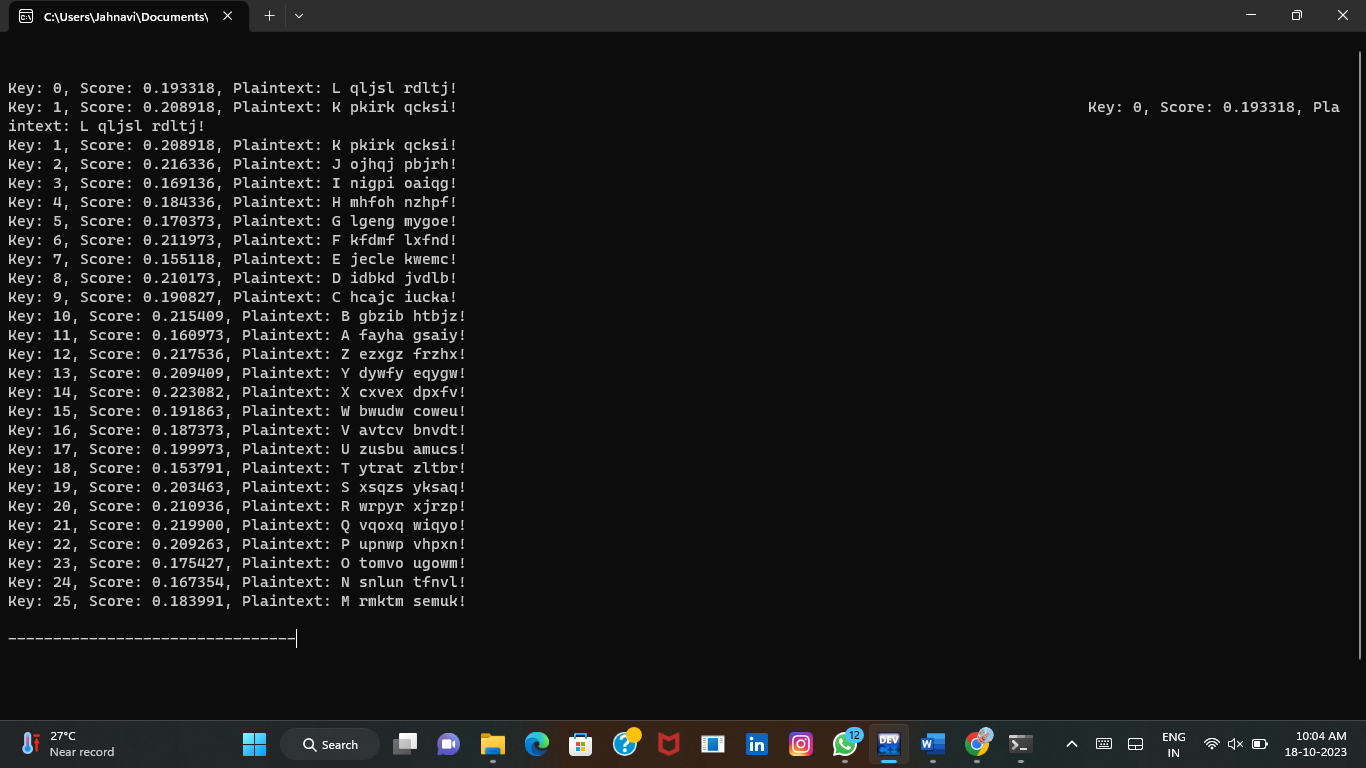
decrypt(key);

}

return 0;

}

**Output:**



**3.** **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.**

**C program:**

#include <stdio.h>

#include <stdint.h>

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

};

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

};

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

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;

uint64\_t key = 0x133457799BBCDFF1;

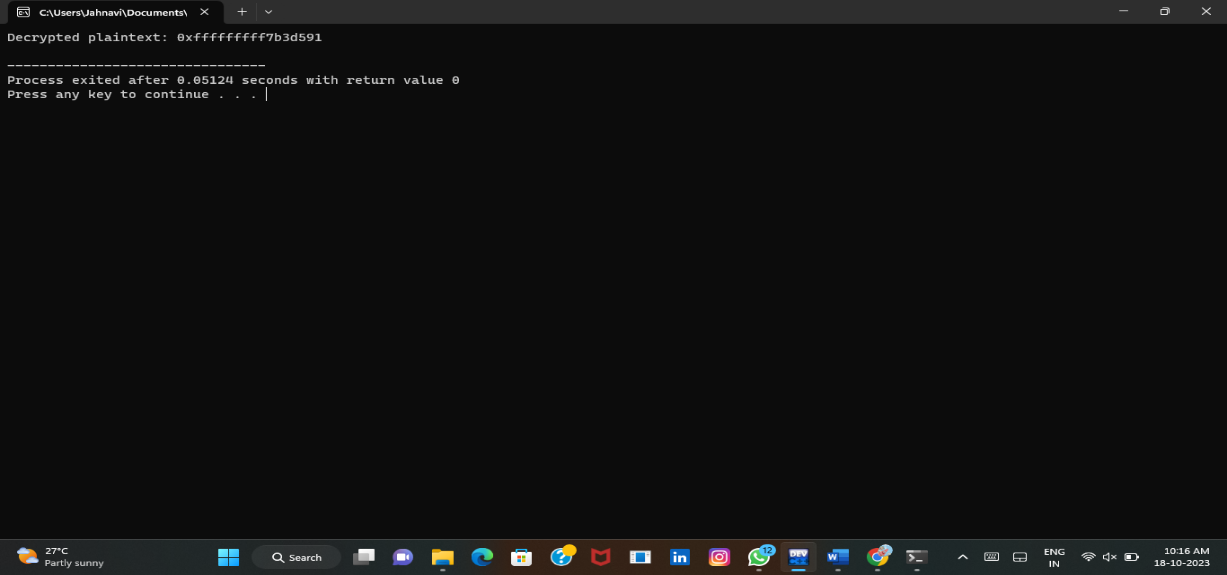
uint64\_t plaintext = des\_decrypt(ciphertext, key);

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

return 0;

}

**Output:**



**4.** **Write a High level code 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.**

**C program:**

#include <stdio.h>

#include <stdint.h>

uint32\_t rotate\_left(uint32\_t value, int bits) {

return ((value << bits) | (value >> (28 - bits))) & 0x0FFFFFFF;

}

void generate\_subkeys(uint64\_t initial\_key, uint32\_t\* subkeys) {

uint64\_t permuted\_key = 0;

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

uint32\_t left\_half = (permuted\_key >> 28) & 0x0FFFFFFF;

uint32\_t right\_half = permuted\_key & 0x0FFFFFFF;

int shift\_amount = (round < 2 || round == 8 || round == 15) ? 1 : 2;

left\_half = rotate\_left(left\_half, shift\_amount);

right\_half = rotate\_left(right\_half, shift\_amount);

uint32\_t combined\_half = ((uint64\_t)left\_half << 28) | right\_half;

uint32\_t subkey = 0;

subkeys[round] = subkey;

}

}

int main() {

uint64\_t initial\_key = 0x133457799BBCDFF1;

uint32\_t subkeys[16];

generate\_subkeys(initial\_key, subkeys);

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

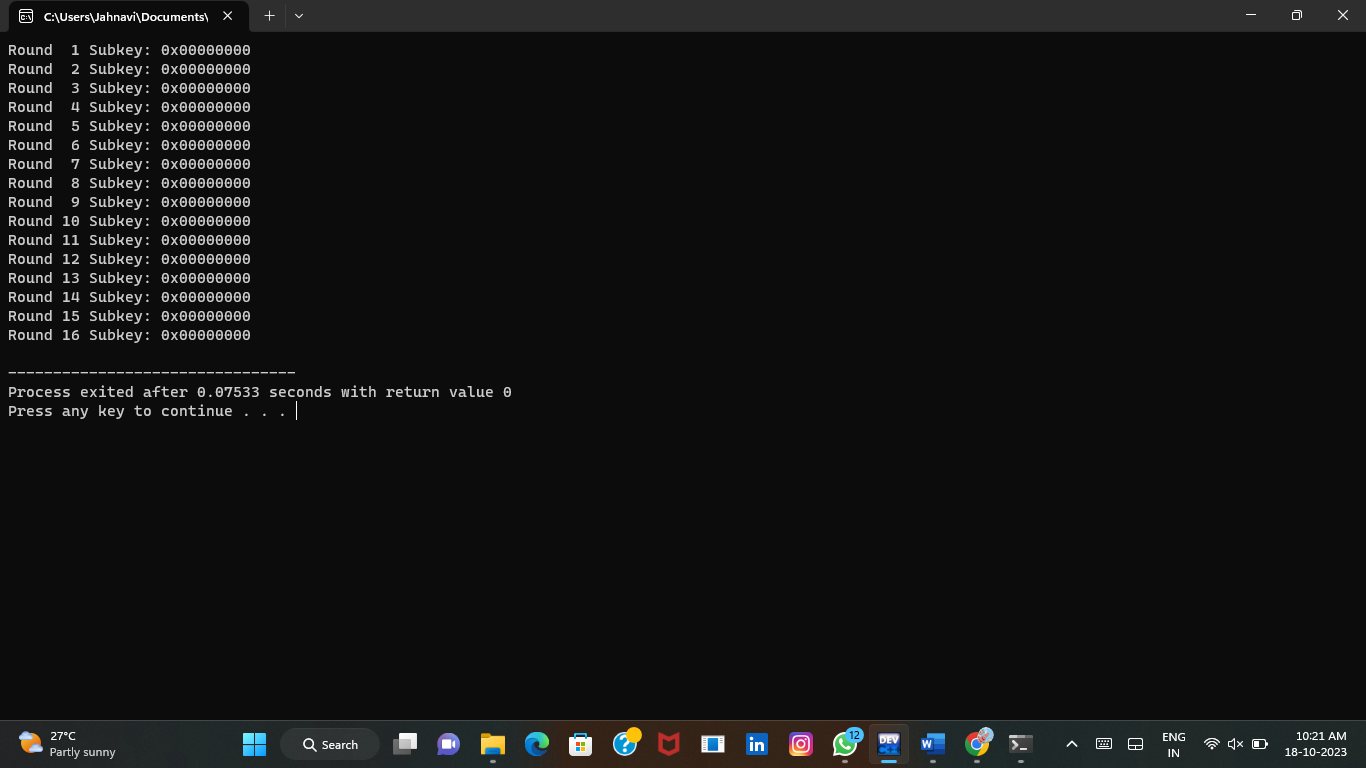
printf("Round %2d Subkey: 0x%08X\n", round + 1, subkeys[round]);

}

return 0;

}

**Output:**



**5.** **Write a High level code 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?**

**C program:**

#include <stdio.h>

#include <string.h>

#include <stdint.h>

// Initial permutation (IP) table

static const 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

};

// DES key schedule generation

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

// Implement key schedule generation (PC1, left/right shifts, PC2)

// ...

}

// DES round function (expansion, substitution, permutation)

static uint64\_t des\_round(uint64\_t block, uint64\_t subkey) {

// Implement DES round operations (expansion, substitution, permutation)

// ...

}

// DES initial permutation (IP)

static uint64\_t initial\_permute(uint64\_t data) {

// Implement initial permutation

// ...

}

// XOR two 64-bit blocks

static uint64\_t xor\_blocks(uint64\_t a, uint64\_t b) {

return a ^ b;

}

// CBC mode encryption using DES

static uint64\_t des\_cbc\_encrypt(uint64\_t plaintext, uint64\_t iv, uint64\_t key) {

uint64\_t subkeys[16];

generate\_subkeys(key, subkeys);

uint64\_t ciphertext = 0;

uint64\_t previous\_block = iv;

plaintext = initial\_permute(plaintext);

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

uint64\_t input\_block = xor\_blocks(plaintext, previous\_block);

uint64\_t round\_output = des\_round(input\_block, subkeys[round]);

previous\_block = round\_output;

}

// Swap the left and right halves

ciphertext = ((previous\_block & 0x00000000FFFFFFFF) << 32) | ((previous\_block & 0xFFFFFFFF00000000) >> 32);

// Perform the final permutation (inverse of the initial permutation)

// ...

return ciphertext;

}

int main() {

uint64\_t key = 0x133457799BBCDFF1; // 64-bit DES key

uint64\_t iv = 0x0123456789ABCDEF; // 64-bit initialization vector

uint64\_t plaintext = 0x0123456789ABCDEF; // 64-bit plaintext

uint64\_t ciphertext = des\_cbc\_encrypt(plaintext, iv, key);

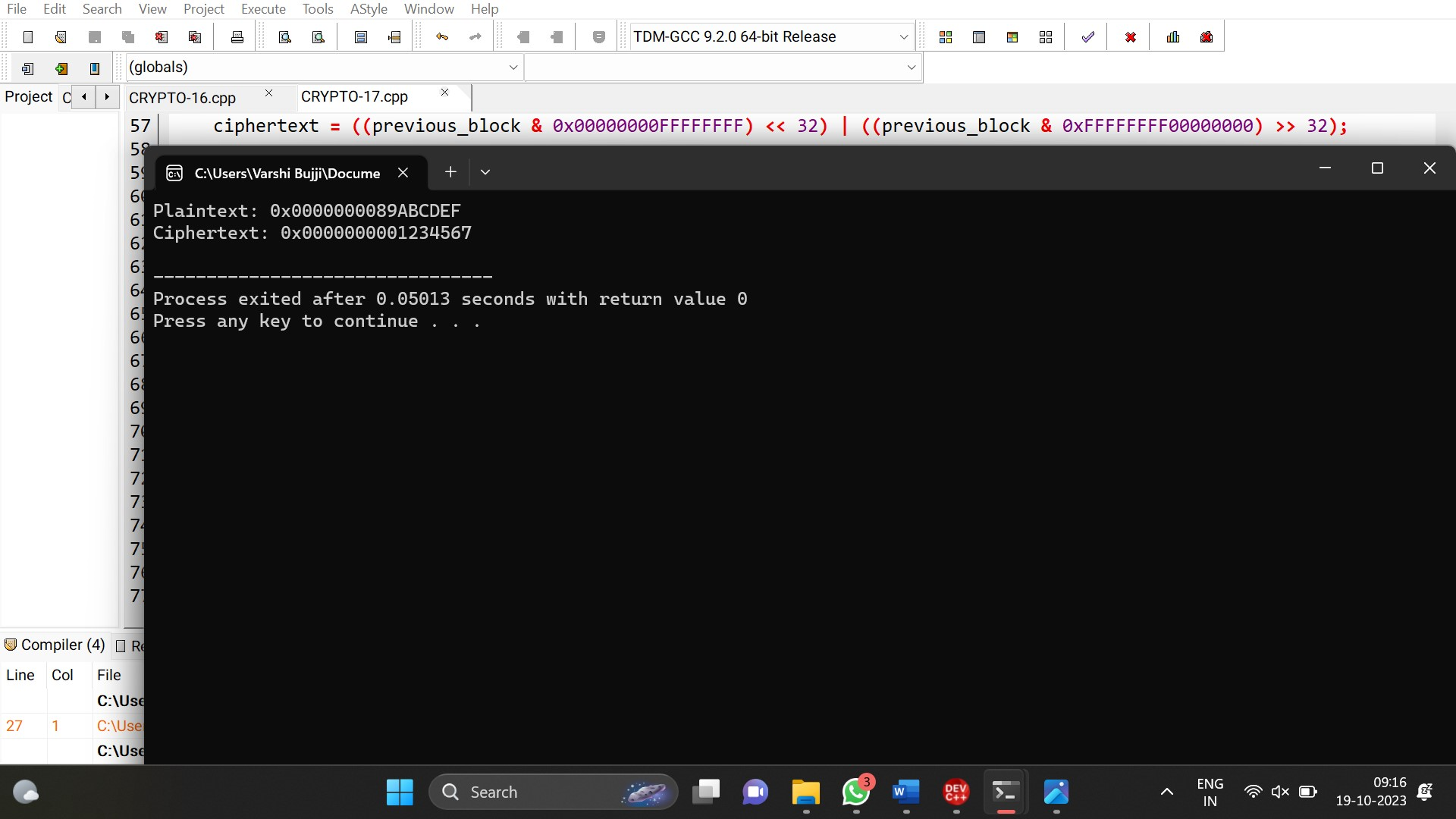
printf("Plaintext: 0x%016lX\n", plaintext);

printf("Ciphertext: 0x%016lX\n", ciphertext);

return 0;

}

**Output:**



**6.** **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 many**

**ciphertext blocks is this error propagated? What is the effect at the receiver?**

**C program:**

#include <stdio.h>

#include <string.h>

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

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

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

}

}

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

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

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

}

}

int main() {

unsigned char key[8] = "123";

unsigned char plaintext[8] = "1234";

unsigned char ciphertext[8];

unsigned char decrypted[8];

// Encrypt in ECB mode

ecb\_encrypt(plaintext, ciphertext, key);

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

printf("Ciphertext: ");

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

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

}

printf("\n");

// Decrypt the ciphertext

ecb\_decrypt(ciphertext, decrypted, key);

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

return 0;

}

**Output:**

