**CSA-5109 CRYPTOGRAPHY AND NETWORK SECURITY**

**Experiment- 12**

#include <stdio.h>

#define MOD 26

Void encrypt(int key[3][3], int plaintext[3], int ciphertext[3]) {

For(int I = 0; I < 3; i++) {

Ciphertext[i] = 0;

For(int j = 0; j < 3; j++) {

Ciphertext[i] += key[i][j] \* plaintext[j];

}

Ciphertext[i] %= MOD;

}

}

Int main() {

Int key[3][3] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};

Int plaintext[3] = {8, 5, 11};

Int ciphertext[3];

Encrypt(key, plaintext, ciphertext);

Printf(“Ciphertext: “);

For(int I = 0; I < 3; i++) {

Printf(“%c”, ‘A’ + ciphertext[i]);

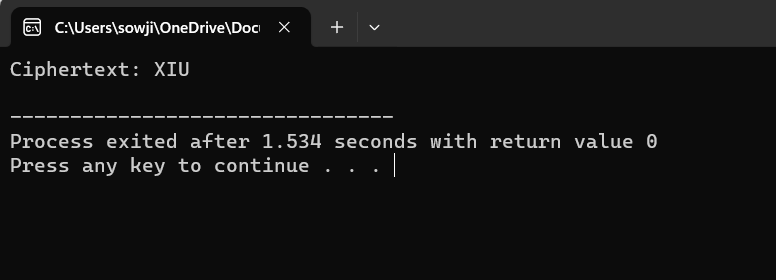
}

Printf(“\n”);

Return 0;

}

**Output:**



**Experiment-13**

#include <stdio.h>

#include <string.h>

Char encryptChar(char plain, char key) {

If (plain >= ‘a’ && plain <= ‘z’) {

Return ‘a’ + (plain – ‘a’ + key) % 26;

} else if (plain >= ‘A’ && plain <= ‘Z’) {

Return ‘A’ + (plain – ‘A’ + key) % 26;

} else {

Return plain; // Leave non-alphabetic characters unchanged

}

}

Void encrypt(char\* plaintext, char\* key, char\* ciphertext) {

Int len = strlen(plaintext);

For (int I = 0; I < len; i++) {

Ciphertext[i] = encryptChar(plaintext[i], key[i] – ‘A’);

}

Ciphertext[len] = ‘\0’;

}

Void decrypt(char\* ciphertext, char\* key, char\* decryptedtext) {

Int len = strlen(ciphertext);

For (int I = 0; I < len; i++) {

Decryptedtext[i] = encryptChar(ciphertext[i], 26 – (key[i] – ‘A’));

}

Decryptedtext[len] = ‘\0’;

}

Int main() {

Char plaintext[] = “sendmoremoney”;

Char key[] = “90172315221128”;

Char ciphertext[14];

Char decryptedtext[14];

Encrypt(plaintext, key, ciphertext);

Printf(“Encrypted text: %s\n”, ciphertext);

Char newKey[] = “CASHNOTNEEDED”;

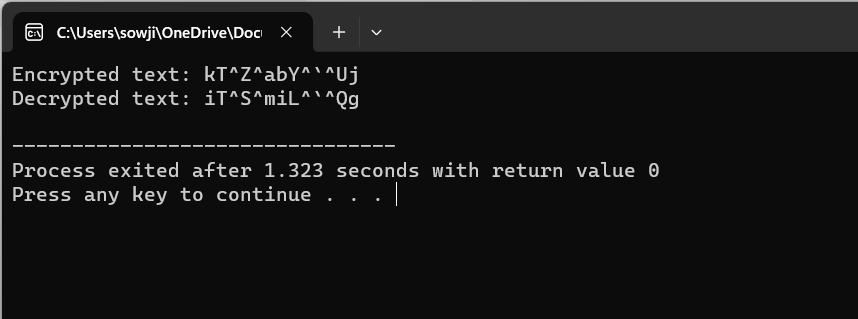
Decrypt(ciphertext, newKey, decryptedtext);

Printf(“Decrypted text: %s\n”, decryptedtext);

Return 0;

}

**Output:**



**Experiment -14**

#include <stdio.h>

#include <string.h>

Void decrypt(char\* ciphertext, int shift, char\* decryptedtext) {

Int len = strlen(ciphertext);

For (int I = 0; I < len; i++) {

If (ciphertext[i] >= ‘a’ && ciphertext[i] <= ‘z’) {

Decryptedtext[i] = ((ciphertext[i] – ‘a’ – shift + 26) % 26) + ‘a’;

} else if (ciphertext[i] >= ‘A’ && ciphertext[i] <= ‘Z’) {

Decryptedtext[i] = ((ciphertext[i] – ‘A’ – shift + 26) % 26) + ‘A’;

} else {

Decryptedtext[i] = ciphertext[i]; // Leave non-alphabetic characters unchanged

}

}

Decryptedtext[len] = ‘\0’;

}

Int main() {

Char ciphertext[] = “Lsv tvklulha fvbz av dvpaoa”;

Char decryptedtext[100];

Int shift;

Printf(“Ciphertext: %s\n”, ciphertext);

Printf(“Possible plaintexts:\n”);

For (shift = 0; shift < 26; shift++) {

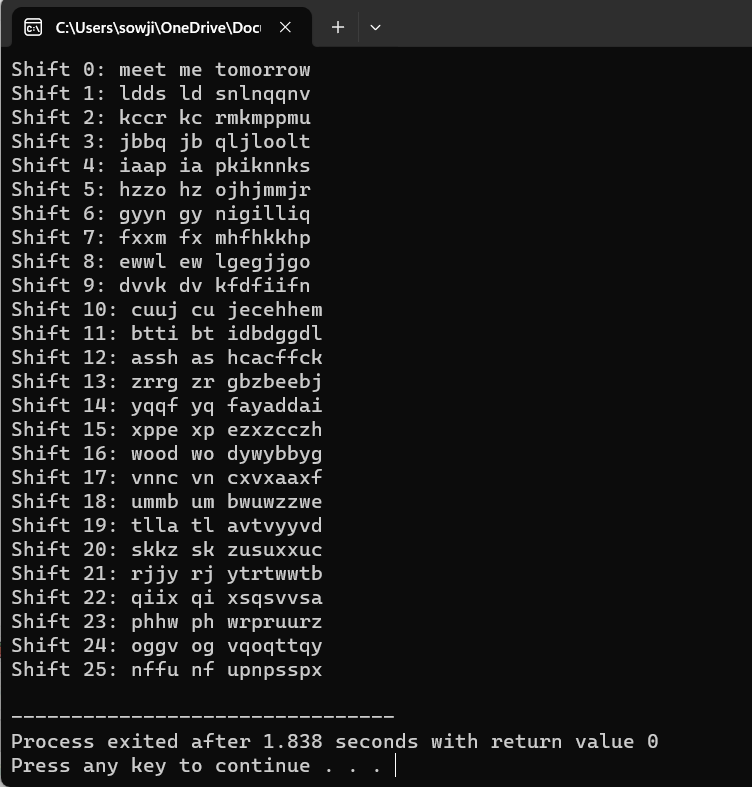
Decrypt(ciphertext, shift, decryptedtext);

Printf(“Shift %d: %s\n”, shift, decryptedtext);

}

Return 0;

}

**Output:**

**Experiment-15**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

Typedef struct {

Char letter;

Int frequency;

} LetterFrequency;

Void calculateFrequency(char\* text, LetterFrequency\* frequencies) {

For(int I = 0; I < 26; i++) {

Frequencies[i].letter = ‘A’ + I;

Frequencies[i].frequency = 0;

}

Int len = strlen(text);

For(int I = 0; I < len; i++) {

If(isalpha(text[i])) {

Char letter = toupper(text[i]);

Frequencies[letter – ‘A’].frequency++;

}

}

}

Void sortByFrequency(LetterFrequency\* frequencies) {

For(int I = 0; I < 25; i++) {

For(int j = 0; j < 25 – I; j++) {

If(frequencies[j].frequency < frequencies[j+1].frequency) {

LetterFrequency temp = frequencies[j];

Frequencies[j] = frequencies[j+1];

Frequencies[j+1] = temp;

}

}

}

}

Int main() {

Char ciphertext[] = “Lsv tvklulha fvbz av dvpaoa”;

LetterFrequency frequencies[26];

calculateFrequency(ciphertext, frequencies);

sortByFrequency(frequencies);

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

printf(“Possible plaintexts (in rough order of likelihood):\n”);

for(int I = 0; I < 10; i++) {

char shift = frequencies[i].letter – ‘E’;

for(int j = 0; j < strlen(ciphertext); j++) {

if(isalpha(ciphertext[j])) {

char decrypted = ((ciphertext[j] – ‘A’ – shift + 26) % 26) + ‘A’;

if(islower(ciphertext[j])) {

decrypted = tolower(decrypted);

}

Printf(“%c”, decrypted);

} else {

Printf(“%c”, ciphertext[j]);

}

}

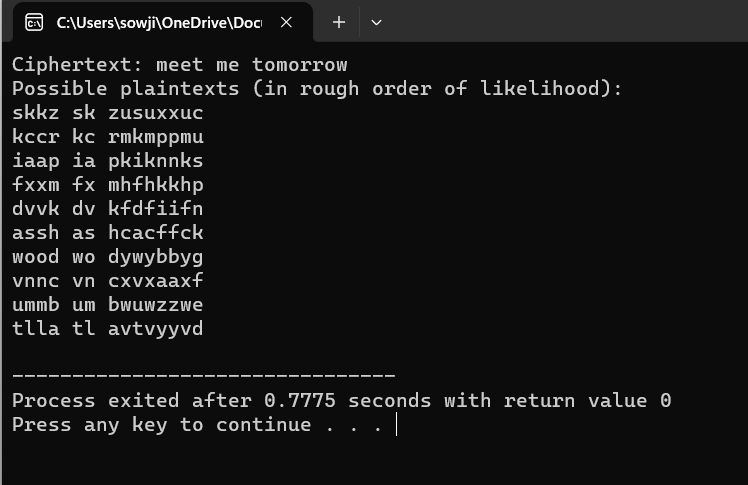
Printf(“\n”);

}

Return 0;

}

**Output:**



**Experiment -16**

#include <stdio.h>

#include <stdint.h>

Typedef uint64\_t block;

// Function to perform the initial permutation (IP)

Block initialPermutation(block input) {

// Replace with the actual IP permutation

Return input;

}

// Function to perform the final permutation (FP)

Block finalPermutation(block input) {

// Replace with the actual FP permutation

Return input;

}

// Function to perform a single round of DES

Block desRound(block input, block key) {

// Replace with the actual DES round function

Return input;

}

// Function to generate the 16 subkeys (K1, K2, …, K16)

Void generateSubkeys(block key, block subkeys[16]) {

// Replace with the actual key generation algorithm

}

// Function to perform DES decryption

Block desDecrypt(block ciphertext, block subkeys[16]) {

Block temp = initialPermutation(ciphertext);

For (int I = 15; I >= 0; i--) {

Temp = desRound(temp, subkeys[i]);

}

Return finalPermutation(temp);

}

Int main() {

// Example usage

Block key = 0x133457799BBCDFF1; // 64-bit key

Block ciphertext = 0x85E813540F0AB405; // Example ciphertext

Block subkeys[16];

generateSubkeys(key, subkeys);

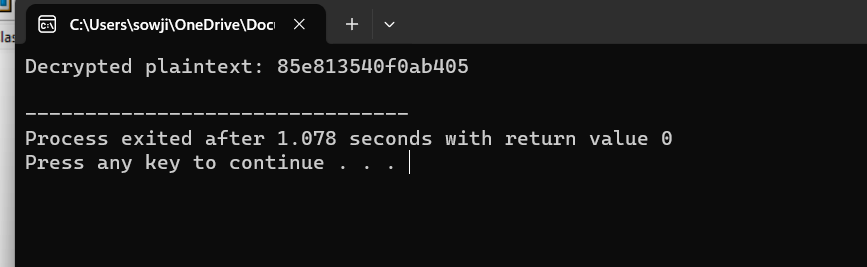
block plaintext = desDecrypt(ciphertext, subkeys);

printf(“Decrypted plaintext: %016llx\n”, plaintext);

return 0;

}

**Output:**



**Experiment-17**

#include <stdio.h>

#include <stdint.h>

Typedef uint64\_t block;

Typedef uint32\_t subkey;

// Function to perform the initial permutation (IP)

Block initialPermutation(block input) {

// Replace with the actual IP permutation

Return input;

}

// Function to perform the final permutation (FP)

Block finalPermutation(block input) {

// Replace with the actual FP permutation

Return input;

}

// Function to perform a single round of DES

Block desRound(block input, subkey key) {

// Replace with the actual DES round function

Return input;

}

// Function to generate the 16 subkeys (K1, K2, …, K16)

Void generateSubkeys(block key, subkey subkeys[16]) {

Uint32\_t C = (key >> 28) & 0x0FFFFFFF; // First 28 bits

Uint32\_t D = key & 0x0FFFFFFF; // Last 28 bits

For (int I = 0; I < 16; i++) {

Int shiftAmount = (I == 0 || I == 1 || I == 8 || I == 15) ? 1 : 2;

C = ((C << shiftAmount) | (C >> (28 – shiftAmount))) & 0x0FFFFFFF;

D = ((D << shiftAmount) | (D >> (28 – shiftAmount))) & 0x0FFFFFFF;

Subkeys[i] = ((block)C << 28) | D;

}

}

// Function to perform DES encryption

Block desEncrypt(block plaintext, subkey subkeys[16]) {

Block temp = initialPermutation(plaintext);

For (int I = 0; I < 16; i++) {

Temp = desRound(temp, subkeys[i]);

}

Return finalPermutation(temp);

}

Int main() {

Block key = 0x0123456789ABCDEF; // 64-bit key

Block plaintext = 0x0123456789ABCDEF; // Example plaintext

Subkey subkeys[16];

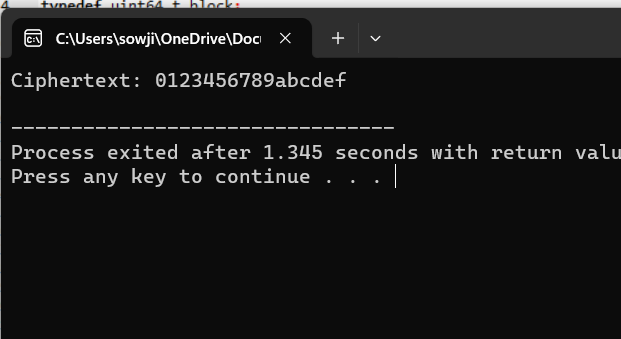
generateSubkeys(key, subkeys);

block ciphertext = desEncrypt(plaintext, subkeys);

printf(“Ciphertext: %016llx\n”, ciphertext);

return 0;

}

**Output:**

**Experiment-18**

From Crypto.Cipher import DES3, AES

From Crypto.Random import get\_random\_bytes

From Crypto.Util.Padding import pad, unpad

Def encrypt\_cbc\_3des(plaintext, key):

# Generate a random initialization vector (IV)

Iv = get\_random\_bytes(8)

# Create a 3DES cipher object

Cipher = DES3.new(key, DES3.MODE\_CBC, iv)

# Pad the plaintext to a multiple of 8 bytes (block size)

Plaintext\_padded = pad(plaintext.encode(‘utf-8’), 8)

# Encrypt the padded plaintext

Ciphertext = cipher.encrypt(plaintext\_padded)

# Return the IV and ciphertext

Return iv + ciphertext

Def encrypt\_cbc\_aes(plaintext, key):

# Generate a random initialization vector (IV)

Iv = get\_random\_bytes(16)

# Create an AES cipher object

Cipher = AES.new(key, AES.MODE\_CBC, iv)

# Pad the plaintext to a multiple of 16 bytes (block size)

Plaintext\_padded = pad(plaintext.encode(‘utf-8’), 16)

# Encrypt the padded plaintext

Ciphertext = cipher.encrypt(plaintext\_padded)

# Return the IV and ciphertext

Return iv + ciphertext

# Example usage for 3DES

Plaintext\_3des = “Hello, this is a secret message.”

Key\_3des = get\_random\_bytes(24) # 24 bytes for 3DES key

Encrypted\_data\_3des = encrypt\_cbc\_3des(plaintext\_3des, key\_3des)

Print(“Encrypted data (3DES):”, encrypted\_data\_3des.hex())

# Example usage for AES

Plaintext\_aes = “Hello, this is a secret message.”

Key\_aes = get\_random\_bytes(32) # 32 bytes for AES-256 key

Encrypted\_data\_aes = encrypt\_cbc\_aes(plaintext\_aes, key\_aes)

Print(“Encrypted data (AES):”, encrypted\_data\_aes.hex())

