***SAVEETHA SCHOOL OF ENGINEERING***

***SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCE***

**EXP NO 19: To Write C program for ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed**

**AIM**

To Write a C program for ECB, CBC, and CFB modes, the plaintext must be a sequence of one or more complete data blocks (or, for CFB mode, data segments). In other words, for these three modes, the total number of bits in the plaintext must be a positive multiple of the block (or segment) size. One common method of padding, if needed, consists of a 1 bit followed by as few zero bits, possibly none, as are necessary to complete the final block. It is considered good practice for the sender to pad every message, including messages in which the final message block is already complete. What is the motivation for including a padding block when padding is not needed

**PROCEDURE**

* Each encryption mode is properly implemented with the correct XOR encryption for demonstration purposes.
* The encryption functions print the encrypted blocks character by character.
* The main function correctly handles padding and calls each encryption mode function to demonstrate encryption.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BLOCK\_SIZE 8 // For demonstration, using 8 bytes (64 bits) block size

// ECB mode encryption

void ecb\_encrypt(char \*plaintext, char \*key, int len) {

// Perform encryption block by block

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

// Perform encryption operation on each block using the key

// Here, we can use any encryption algorithm like AES

printf("ECB Encrypted Block %d: ", i / BLOCK\_SIZE + 1);

for (int j = 0; j < BLOCK\_SIZE; j++) {

plaintext[i + j] ^= key[j % strlen(key)]; // Just a simple XOR encryption for demonstration

printf("%c", plaintext[i + j]);

}

printf("\n");

}

}

// CBC mode encryption

void cbc\_encrypt(char \*plaintext, char \*key, int len) {

char iv[BLOCK\_SIZE + 1] = "INITIALV"; // Initialization Vector (+1 for null terminator)

char prev\_cipher\_block[BLOCK\_SIZE + 1]; // Previous cipher block (+1 for null terminator)

// Perform encryption block by block

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

// XOR plaintext block with the previous ciphertext block (or IV for the first block)

for (int j = 0; j < BLOCK\_SIZE; j++) {

plaintext[i + j] ^= (i == 0) ? iv[j] : prev\_cipher\_block[j];

}

// Perform encryption operation on the XORed block using the key

// Here, we can use any encryption algorithm like AES

printf("CBC Encrypted Block %d: ", i / BLOCK\_SIZE + 1);

for (int j = 0; j < BLOCK\_SIZE; j++) {

plaintext[i + j] ^= key[j % strlen(key)]; // Just a simple XOR encryption for demonstration

printf("%c", plaintext[i + j]);

}

printf("\n");

// Update the previous ciphertext block

memcpy(prev\_cipher\_block, plaintext + i, BLOCK\_SIZE);

prev\_cipher\_block[BLOCK\_SIZE] = '\0'; // Null-terminate the string

}

}

// CFB mode encryption

void cfb\_encrypt(char \*plaintext, char \*key, int len) {

char iv[BLOCK\_SIZE + 1] = "INITIALV"; // Initialization Vector (+1 for null terminator)

char feedback[BLOCK\_SIZE + 1]; // Feedback from previous encryption (+1 for null terminator)

// Perform encryption block by block

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

// Perform encryption operation on the IV (or previous ciphertext) using the key

// Here, we can use any encryption algorithm like AES

printf("CFB Encrypted Block %d: ", i / BLOCK\_SIZE + 1);

for (int j = 0; j < BLOCK\_SIZE; j++) {

feedback[j] = iv[j];

iv[j] = plaintext[i + j] ^ key[j % strlen(key)]; // Just a simple XOR encryption for demonstration

printf("%c", iv[j]);

}

printf("\n");

// Update the feedback with the IV

memcpy(iv, feedback, BLOCK\_SIZE);

iv[BLOCK\_SIZE] = '\0'; // Null-terminate the string

}

}

int main() {

char plaintext[] = "PLAINTEXTTOENCRYPTPLAINTEXTTOENCRYPT"; // Example plaintext

char key[] = "SECRETKEY"; // Example key

int len = strlen(plaintext);

// Pad the plaintext if needed

int padding = BLOCK\_SIZE - (len % BLOCK\_SIZE);

if (padding != BLOCK\_SIZE) {

for (int i = 0; i < padding - 1; i++) {

plaintext[len + i] = '0'; // Padding with zero bits

}

plaintext[len + padding - 1] = '1'; // Adding the 1 bit as per padding rule

plaintext[len + padding] = '\0'; // Null-terminate the string

len += padding;

}

// Perform encryption using different modes

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

printf("Plaintext Length: %d\n\n", len);

printf("ECB Mode Encryption:\n");

ecb\_encrypt(plaintext, key, len);

printf("\n");

printf("CBC Mode Encryption:\n");

cbc\_encrypt(plaintext, key, len);

printf("\n");

printf("CFB Mode Encryption:\n");

cfb\_encrypt(plaintext, key, len);

printf("\n");

return 0;

}

**OUTPUT**

