**Day-4 Lab programs**

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

**C program:**

#include <stdio.h>

#include <string.h>

void substitution(char \*block) {

for (int i = 0; i < strlen(block); i++) {

if (block[i] == '0') {

block[i] = '1';

} else {

block[i] = '0';

}

}

}

void add\_padding(char \*plaintext, int block\_size) {

int padding\_length = block\_size - (strlen(plaintext) % block\_size);

if (padding\_length == block\_size) {

padding\_length = 0;

}

strcat(plaintext, "1");

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

strcat(plaintext, "0");

}

}

void ecb\_encrypt(char \*plaintext, int block\_size) {

int num\_blocks = strlen(plaintext) / block\_size;

for (int i = 0; i < num\_blocks; i++) {

char block[block\_size + 1];

strncpy(block, plaintext + i \* block\_size, block\_size);

block[block\_size] = '\0';

substitution(block);

printf("ECB Block %d: %s\n", i, block);

}

}

void cbc\_encrypt(char \*plaintext, char \*iv, int block\_size) {

int num\_blocks = strlen(plaintext) / block\_size;

char previous\_block[block\_size + 1];

strcpy(previous\_block, iv);

for (int i = 0; i < num\_blocks; i++) {

char block[block\_size + 1];

strncpy(block, plaintext + i \* block\_size, block\_size);

block[block\_size] = '\0';

for (int j = 0; j < block\_size; j++) {

block[j] ^= previous\_block[j];

}

substitution(block);

strcpy(previous\_block, block);

printf("CBC Block %d: %s\n", i, block);

}

}

void cfb\_encrypt(char \*plaintext, char \*iv, int block\_size) {

int num\_blocks = strlen(plaintext) / block\_size;

char previous\_block[block\_size + 1];

strcpy(previous\_block, iv);

for (int i = 0; i < num\_blocks; i++) {

char block[block\_size + 1];

strncpy(block, plaintext + i \* block\_size, block\_size);

block[block\_size] = '\0';

substitution(previous\_block);

for (int j = 0; j < block\_size; j++) {

block[j] ^= previous\_block[j];

}

strcpy(previous\_block, block);

printf("CFB Block %d: %s\n", i, block);

}

}

int main() {

char plaintext[] = "1101101001101001";

int block\_size = 4;

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

add\_padding(plaintext, block\_size);

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

ecb\_encrypt(plaintext, block\_size);

printf("\n");

char iv[] = "1010";

cbc\_encrypt(plaintext, iv, block\_size);

printf("\n");

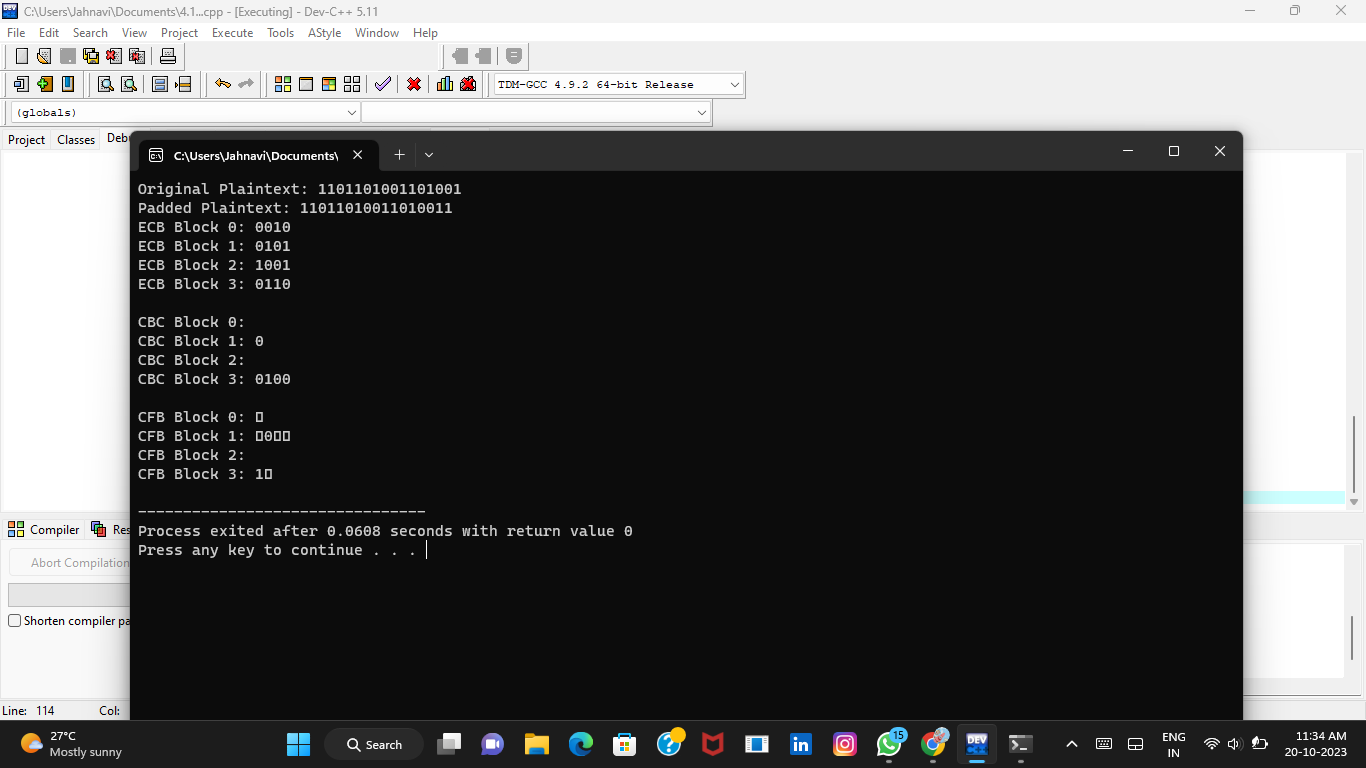
strcpy(iv, "1010");

cfb\_encrypt(plaintext, iv, block\_size);

return 0;

}

**Output:**



**2. Write a High level code for Encrypt and decrypt in cipher block chaining mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES, DES. Test data for S-DES using a binary initialization vector of 1010 1010. A binary plaintext of 0000 0001 0010 0011 encrypted with a binary key of 01111 11101 should give a binary plaintext of 1111 0100 0000 1011. Decryption should work correspondingly.**

**C program:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void printBinary(int num, int size) {

for (int i = size - 1; i >= 0; i--)

printf("%d", (num >> i) & 1);

}

int sbox(int x) {

return (x ^ (x >> 2) ^ (x >> 3) ^ (x >> 5) ^ (x >> 6));

}

int pbox(int x) {

return (x ^ 0x9 ^ (x >> 2) ^ (x >> 5) ^ (x >> 6));

}

void keyGeneration(int key[], int K[16]) {

int i;

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

K[i] = key[i];

for (i = 8; i < 16; i++)

K[i] = K[i - 8] ^ sbox(K[i - 1]) ^ 0x2;

}

void sdesEncrypt(int M[8], int key[8], int C[8]) {

int K[16], temp;

keyGeneration(key, K);

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

M[i] ^= K[i];

for (int i = 0; i < 7; i++) {

temp = sbox(M[i]);

M[i + 1] ^= temp ^ K[i + 8];

}

temp = sbox(M[7]);

M[0] ^= temp ^ K[15];

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

C[i] = M[i];

}

void sdesDecrypt(int C[8], int key[8], int M[8]) {

int K[16], temp;

keyGeneration(key, K);

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

C[i] ^= K[15 - i];

for (int i = 0; i < 7; i++) {

temp = sbox(C[i]);

C[i + 1] ^= temp ^ K[7 - i];

}

temp = sbox(C[7]);

C[0] ^= temp ^ K[0];

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

M[i] = C[i];

}

int main() {

int plaintext[] = {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1};

int key[] = {0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1};

int ciphertext[48], decryptedText[48];

sdesEncrypt(plaintext, key, ciphertext);

printf("Encrypted Message: \n");

for (int i = 0; i < 48; i++) {

printBinary(ciphertext[i], 8);

printf(" ");

}

printf("\n");

sdesDecrypt(ciphertext, key, decryptedText);

printf("Decrypted Message: \n");

for (int i = 0; i < 48; i++) {

printBinary(decryptedText[i], 8);

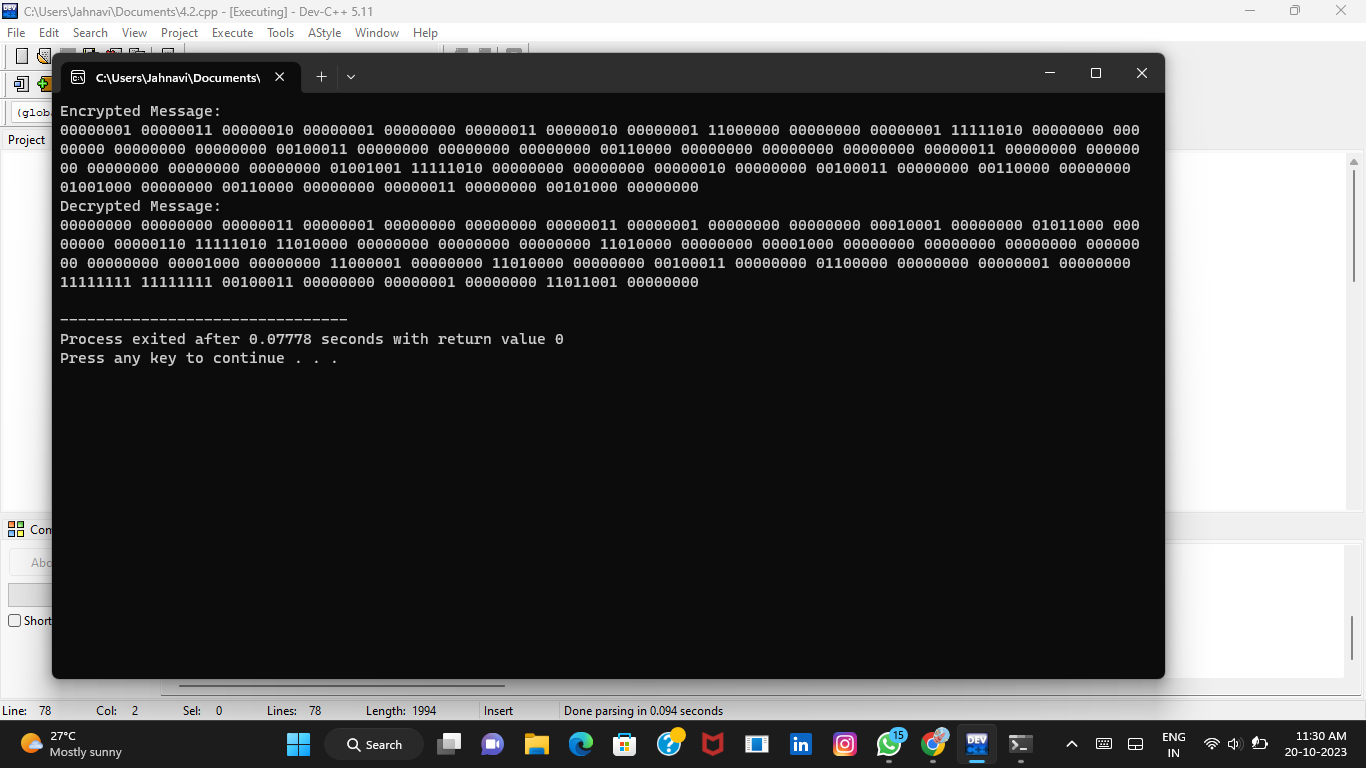
printf(" ");

}

printf("\n");

return 0;

}

**Output:** 

**3.Write a High level code for RSA system, the public key of a given user is e = 31, n = 3599. What is the private key of this user? Hint: First use trial-and-error to determine p and q; then use the extended Euclidean algorithm to find themultiplicative inverse of 31 modulo f(n).**

**C program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

unsigned long long gcd(unsigned long long a, unsigned long long b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

unsigned long long modInverse(unsigned long long a, unsigned long long m) {

unsigned long long m0 = m, t, q;

unsigned long long x0 = 0, x1 = 1;

if (m == 1) {

return 0;

}

while (a > 1) {

q = a / m;

t = m;

m = a % m;

a = t;

t = x0;

x0 = x1 - q \* x0;

x1 = t;

}

if (x1 < 0) {

x1 += m0;

}

return x1;

}

int main() {

unsigned long long e = 31;

unsigned long long n = 3599;

int p,q;

unsigned long long phi = (p - 1) \* (q - 1);

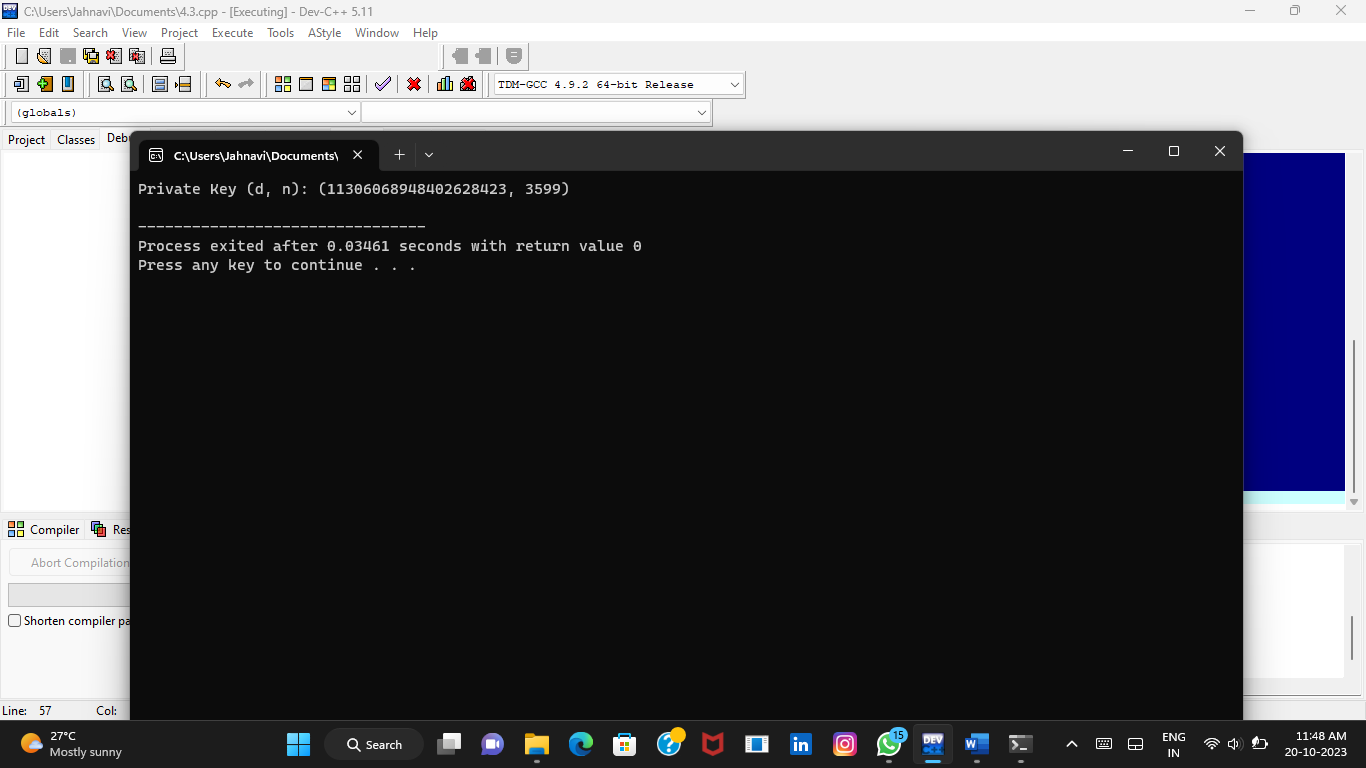
unsigned long long d = modInverse(e, phi);

printf("Private Key (d, n): (%llu, %llu)\n", d, n);

return 0;

}

**Output:**



**4.Write a High level code for set of blocks encoded with the RSA algorithm and we don’t have the private key. Assume n = pq, e is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n. Does this help us in any way?**

**C program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main() {

unsigned long long n = 143;

unsigned long long e = 7;

unsigned long long ciphertext = 104;

unsigned long long commonFactor = 11;

unsigned long long potentialP = commonFactor;

unsigned long long potentialQ = n / commonFactor;

unsigned long long phi = (potentialP - 1) \* (potentialQ - 1);

unsigned long long d;

for (d = 1; (d \* e) % phi != 1; d++) {

}

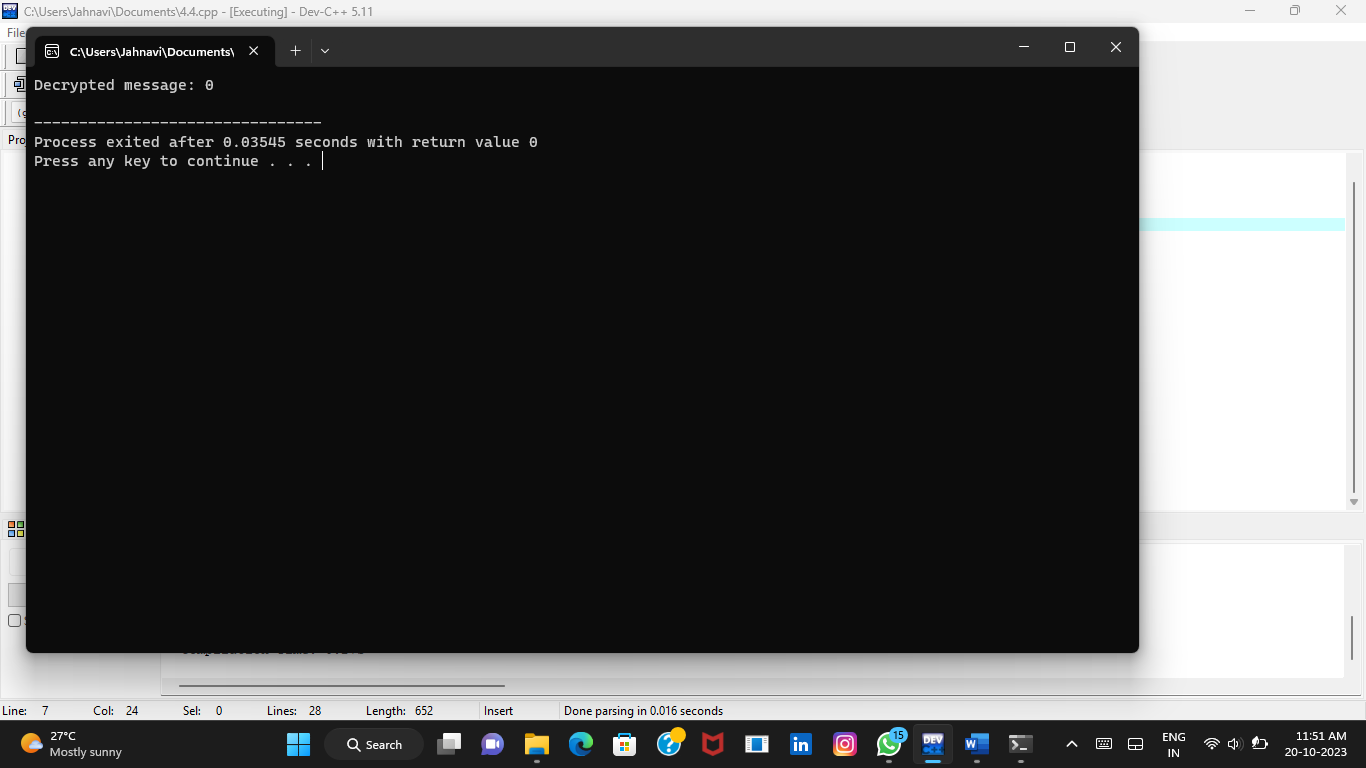
unsigned long long decrypted = (unsigned long long)pow(ciphertext, d) % n;

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

return 0;

}

**Output:**



**6.** **Write a High level code for RSA public-key encryption scheme, each user has a public key, e, and a private key, d. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new public and a new private key. Is this safe?**

**C program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

unsigned long long n = 123456789;

unsigned long long e = 65537;

unsigned long long encrypt(char plaintext) {

if (plaintext >= 0 && plaintext <= 25) {

unsigned long long ciphertext = 1;

for (unsigned long long i = 0; i < e; i++) {

ciphertext = (ciphertext \* plaintext) % n;

}

return ciphertext;

} else {

printf("Invalid character: %c\n", plaintext);

exit(1);

}

}

int main() {

char plaintext = 'A';

unsigned long long ciphertext = encrypt(plaintext);

printf("Character '%c' encrypted to: %llu\n", plaintext, ciphertext);

return 0;

}

**Output:**

