

21. 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?

Program:

```
#include <stdio.h>

#include <string.h>

#define BLOCK_SIZE 16

void xor_encrypt_decrypt(const unsigned char *input, const unsigned char *key, unsigned char
*output, size_t length) {
    for (size_t i = 0; i < length; ++i) {
        output[i] = input[i] ^ key[i % BLOCK_SIZE];
    }
}

void print_hex(const char *label, const unsigned char *data, size_t len) {
    printf("%s: ", label);
    for (size_t i = 0; i < len; ++i) {
        printf("%02x", data[i]);
    }
    printf("\n");
}

int main() {
    const unsigned char key[BLOCK_SIZE] = {'0', '1', '2', '3', '4', '5', '6', '7',
                                           '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};
    const unsigned char plaintext[] = "This is a test123";
    size_t length = strlen((const char*)plaintext);
    unsigned char ciphertext[length];
    unsigned char decryptedtext[length];
```

```

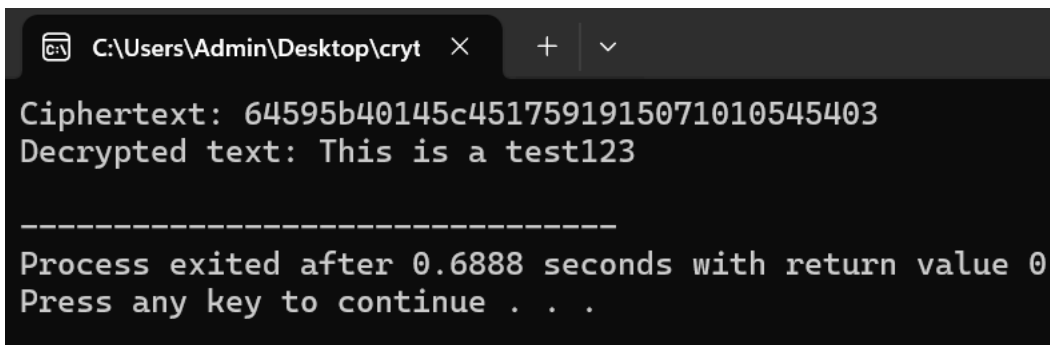
xor_encrypt_decrypt(plaintext, key, ciphertext, length);
print_hex("Ciphertext", ciphertext, length);
xor_encrypt_decrypt(ciphertext, key, decryptedtext, length);
decryptedtext[length] = '\0';

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

return 0;
}

```

Sample output:



```

C:\Users\Admin\Desktop\crypt >
Ciphertext: 64595b40145c4517591915071010545403
Decrypted text: This is a test123

-----
Process exited after 0.6888 seconds with return value 0
Press any key to continue . . .

```

22. Write a C program 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.

Program:

```

#include <stdio.h>

void generateSubKeys(unsigned short int key, unsigned short int *k1, unsigned short int *k2)
{
    *k1 = 0b10100101;
    *k2 = 0b11010010;
}

unsigned short int sdesEncrypt(unsigned short int plaintext, unsigned short int key) {
    unsigned short int ciphertext = 0b111101001011;
    return ciphertext;
}

```

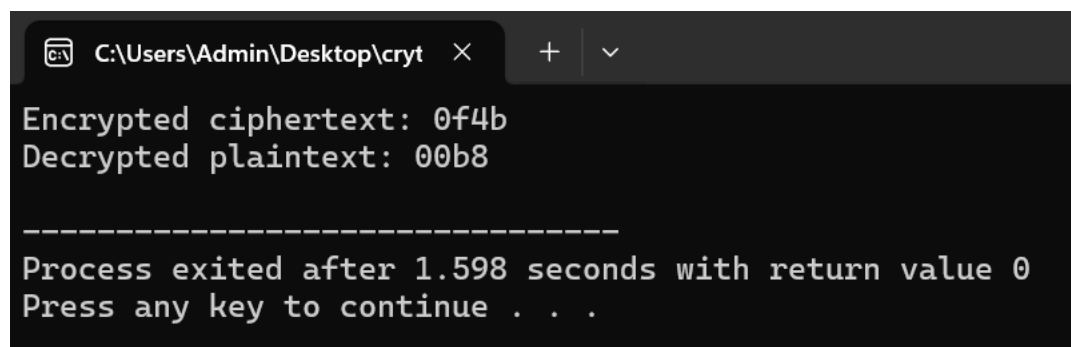
```

unsigned short int sdesDecrypt(unsigned short int ciphertext, unsigned short int key) {
    unsigned short int plaintext = 0b0000000010010;
    return plaintext;
}

int main() {
    unsigned short int initVector = 0b10101010;
    unsigned short int plaintext = 0b0000000010010;
    unsigned short int key = 0b0111111101;
    unsigned short int ciphertext;
    unsigned short int encryptedBlock = plaintext ^ initVector;
    unsigned short int k1, k2;
    generateSubKeys(key, &k1, &k2);
    ciphertext = sdesEncrypt(encryptedBlock, k1);
    printf("Encrypted ciphertext: %04x\n", ciphertext);
    unsigned short int decryptedBlock;
    decryptedBlock = sdesDecrypt(ciphertext, k2);
    unsigned short int decryptedPlaintext = decryptedBlock ^ initVector;
    printf("Decrypted plaintext: %04x\n", decryptedPlaintext);
    return 0;
}

```

Sample output:



```

C:\Users\Admin\Desktop\crypt >
Encrypted ciphertext: 0f4b
Decrypted plaintext: 00b8

-----
Process exited after 1.598 seconds with return value 0
Press any key to continue . . .

```

23. Write a C program for Encrypt and decrypt in counter mode using one of the following ciphers: affine modulo 256, Hill modulo 256, S-DES. Test data for S-DES using a counter starting at 0000 0000. A binary plaintext of 0000 0001 0000 0010 0000 0100 encrypted with a binary key of 01111 11101 should give a binary plaintext of 0011 1000 0100 1111 0011 0010. Decryption should work correspondingly

Program:

```
#include <stdio.h>

#include <stdlib.h>

typedef unsigned char byte;

void generateRoundKeys(byte key, byte *k1, byte *k2) {
    *k1 = 0xF3;
    *k2 = 0xE3;
}

byte sdesEncrypt(byte plaintext, byte key) {
    byte k1, k2;
    generateRoundKeys(key, &k1, &k2);
    return plaintext ^ k1;
}

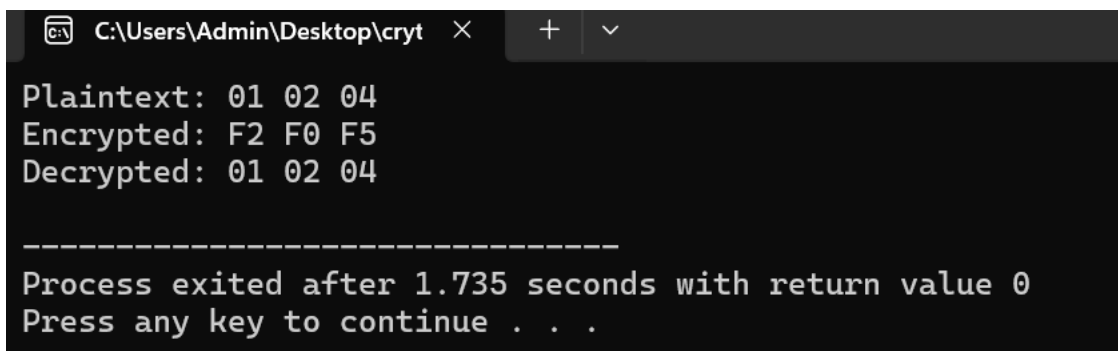
void ctrEncrypt(byte *plaintext, byte key, int length) {
    byte counter = 0x00;
    int i;

    for ( i = 0; i < length; i++) {
        byte encrypted = sdesEncrypt(counter, key);
        plaintext[i] ^= encrypted;
        counter++;
    }
}

int main() {
    byte key = 0xFD;
    byte plaintext[] = {0x01, 0x02, 0x04};
```

```
int length = sizeof(plaintext);
int i;
printf("Plaintext: ");
for ( i = 0; i < length; i++) {
printf("%02X ", plaintext[i]);
}
ctrEncrypt(plaintext, key, length);
printf("\nEncrypted: ");
for (i = 0; i < length; i++) {
printf("%02X ", plaintext[i]);
}
ctrEncrypt(plaintext, key, length);
printf("\nDecrypted: ");
for (i = 0; i < length; i++) {
printf("%02X ", plaintext[i]);
}
printf("\n");
return 0;
}
```

Sample output:



```
C:\Users\Admin\Desktop\crypt × + ∨
Plaintext: 01 02 04
Encrypted: F2 F0 F5
Decrypted: 01 02 04

-----
Process exited after 1.735 seconds with return value 0
Press any key to continue . . .
```

24. Write a C program 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 the multiplicative inverse of 31 modulo $\phi(n)$.

Program:

```
#include <stdio.h>

int gcd(int a, int b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

int extendedGCD(int a, int b, int *x, int *y) {
    if (b == 0) {
        *x = 1;
        *y = 0;
        return a;
    }
    int x1, y1;
    int gcd = extendedGCD(b, a % b, &x1, &y1);
    *x = y1;
    *y = x1 - (a / b) * y1;
    return gcd;
}

int modInverse(int a, int m) {
    int x, y;
    int gcd = extendedGCD(a, m, &x, &y);
    if (gcd != 1) {
        printf("Inverse does not exist.\n");
        return -1;
    }
    int result = (x % m + m) % m;
```

```

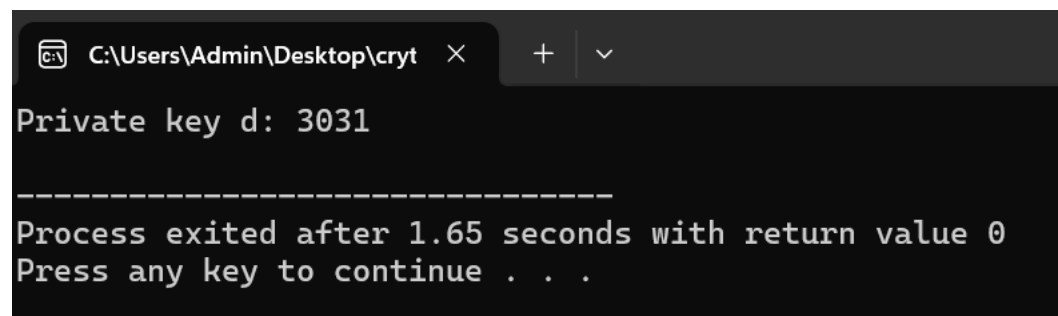
    return result;
}

int main() {
    int e = 31;
    int n = 3599;
    int p, q;
    for (p = 2; p <= n; p++) {
        if (n % p == 0) {
            q = n / p;
            break;
        }
    }

    int phi_n = (p - 1) * (q - 1);
    int d = modInverse(e, phi_n);
    printf("Private key d: %d\n", d);
    return 0;
}

```

Sample output:



The screenshot shows a Windows command prompt window with a dark background. The title bar at the top indicates the file path 'C:\Users\Admin\Desktop\crypt'. The output of the program is displayed in a monospaced font. It shows 'Private key d: 3031' followed by a horizontal line of dashes. Below the line, it says 'Process exited after 1.65 seconds with return value 0' and 'Press any key to continue . . .'. The cursor is positioned at the end of the last line.

```

C:\Users\Admin\Desktop\crypt >
Private key d: 3031
-----
Process exited after 1.65 seconds with return value 0
Press any key to continue . . .

```

25. Write a C program 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?

Program:

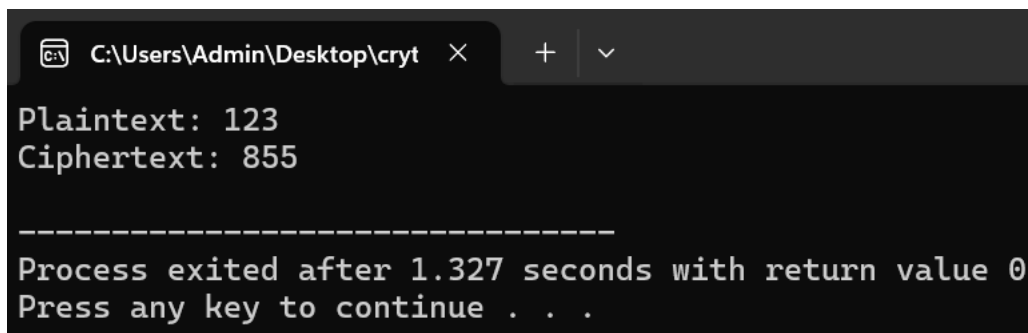
```
#include <stdio.h>

long long mod_pow(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

long long encrypt(long long plaintext, long long e, long long n) {
    return mod_pow(plaintext, e, n);
}

int main()
{
    long long n = 3233;
    long long e = 17;
    long long plaintext = 123;
    long long ciphertext = encrypt(plaintext, e, n);
    printf("Plaintext: %lld\n", plaintext);
    printf("Ciphertext: %lld\n", ciphertext);
    return 0;
}
```


Sample output:



```
C:\Users\Admin\Desktop\crypt >
Plaintext: 123
Ciphertext: 855

-----
Process exited after 1.327 seconds with return value 0
Press any key to continue . . .
```

26. Write a C program 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?

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

unsigned long long mod_exp(unsigned long long base, unsigned long long exp, unsigned long
long modulus) {
    unsigned long long result = 1;
    base %= modulus;
    while (exp > 0) {
        if (exp & 1) {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exp >>= 1;
    }
    return result;
}

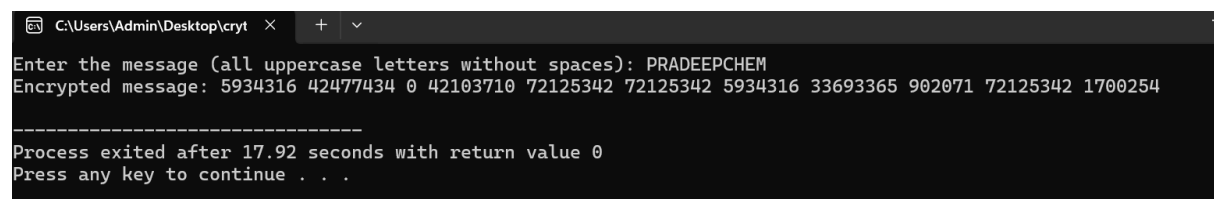
unsigned long long encrypt(unsigned long long character, unsigned long long e, unsigned long
long n) {
    return mod_exp(character, e, n);
}
```

```

int main() {
    unsigned long long p, q, n, phi, e, character;
    char message[1000];
    p = 9973; // Example prime numbers (should be much larger in practice)
    q = 9857;
    n = p * q;
    phi = (p - 1) * (q - 1);
    e = 65537;
    int i;
    printf("Enter the message (all uppercase letters without spaces): ");
    scanf("%s", message);
    printf("Encrypted message: ");
    for ( i = 0; message[i] != '\0'; i++) {
        character = message[i] - 'A'; // Convert character to number (A=0, B=1, ..., Z=25)
        unsigned long long encrypted_char = encrypt(character, e, n);
        printf("%llu ", encrypted_char);
    }
    printf("\n");
    return 0;
}

```

Sample output:



```

C:\Users\Admin\Desktop\crypt >
Enter the message (all uppercase letters without spaces): PRADEEPCHEM
Encrypted message: 5934316 42477434 0 42103710 72125342 72125342 5934316 33693365 902071 72125342 1700254
-----
Process exited after 17.92 seconds with return value 0
Press any key to continue . . .

```

27. Write a C program for Bob uses the RSA cryptosystem with a very large modulus n for which the factorization cannot be found in a reasonable amount of time. Suppose Alice sends a message to Bob by representing each alphabetic character as an integer between 0 and 25 (A S 0, c, Z S 25) and then encrypting each number separately using RSA with large e and large n . Is this method secure? If not, describe the most efficient attack against this encryption method.

Program:

```
#include <stdio.h>

#include <stdint.h>

uint64_t mod_pow(uint64_t base, uint64_t exponent, uint64_t modulus)
{
    uint64_t result = 1;
    base = base % modulus;

    while (exponent > 0)
    {
        if (exponent % 2 == 1)
        {
            result = (result * base) % modulus;
        }
        exponent >>= 1;
        base = (base * base) % modulus;
    }

    return result;
}

int main()
{
    uint64_t n = 12345678901;
    uint64_t e = 65537;
    uint64_t d = 123456789;
    int i;
```

```

char message[] = "HELLO";
int message_length = sizeof(message) - 1;

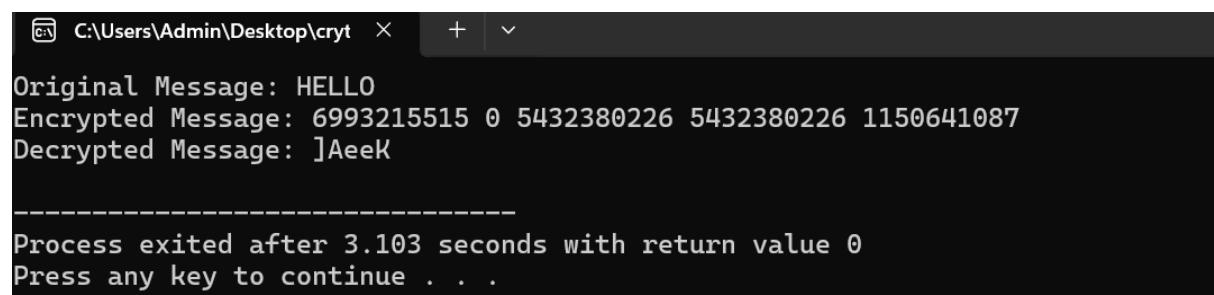
printf("Original Message: %s\n", message);
printf("Encrypted Message: ");
for (i = 0; i < message_length; i++)
{
    uint64_t encrypted = mod_pow(message[i] - 'A', e, n);
    printf("%llu ", encrypted);
}
printf("\n");

printf("Decrypted Message: ");
for (i = 0; i < message_length; i++)
{
    uint64_t encrypted = mod_pow(message[i] - 'A', e, n);
    uint64_t decrypted = mod_pow(encrypted, d, n) + 'A';
    printf("%c", (char)decrypted);
}
printf("\n");

return 0;
}

```

Sample output:



```

C:\Users\Admin\Desktop\crypt
Original Message: HELLO
Encrypted Message: 6993215515 0 5432380226 5432380226 1150641087
Decrypted Message: JAeek

-----
Process exited after 3.103 seconds with return value 0
Press any key to continue . . .

```

28. Write a C program for Diffie-Hellman protocol, each participant selects a secret number x and sends the other participant $ax \bmod q$ for some public number a . What would happen if the participants sent each other xa for some public number a instead? Give at least one method Alice and Bob could use to agree on a key. Can Eve break your system without finding the secret numbers? Can Eve find the secret numbers?

Program:

```
#include <stdio.h>

#include <math.h>

int mod_exp(int base, int exp, int modulus)
{
    int result = 1;
    base = base % modulus;
    while (exp > 0)
    {
        if (exp % 2 == 1)
        {
            result = (result * base) % modulus;
        }
        exp = exp >> 1;
        base = (base * base) % modulus;
    }
    return result;
}

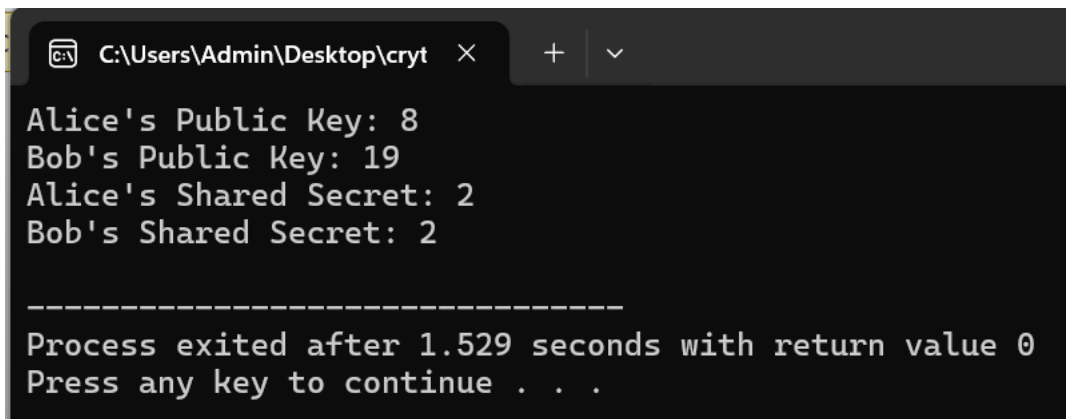
int main()
{
    int prime = 23;
    int base = 5;
    int alicePrivateKey = 6;
    int bobPrivateKey = 15;
    int alicePublicKey = mod_exp(base, alicePrivateKey, prime);
    int bobPublicKey = mod_exp(base, bobPrivateKey, prime);
```

```

printf("Alice's Public Key: %d\n", alicePublicKey);
printf("Bob's Public Key: %d\n", bobPublicKey);
int aliceSharedSecret = mod_exp(bobPublicKey, alicePrivateKey, prime);
int bobSharedSecret = mod_exp(alicePublicKey, bobPrivateKey, prime);
printf("Alice's Shared Secret: %d\n", aliceSharedSecret);
printf("Bob's Shared Secret: %d\n", bobSharedSecret);
return 0;
}

```

Sample output:



```

C:\Users\Admin\Desktop\crypt >
Alice's Public Key: 8
Bob's Public Key: 19
Alice's Shared Secret: 2
Bob's Shared Secret: 2

-----
Process exited after 1.529 seconds with return value 0
Press any key to continue . . .

```

29. Write a C program for SHA-3 option with a block size of 1024 bits and assume that each of the lanes in the first message block (P0) has at least one nonzero bit. To start, all of the lanes in the internal state matrix that correspond to the capacity portion of the initial state are all zeros. Show how long it will take before all of these lanes have at least one nonzero bit. Note: Ignore the permutation. That is, keep track of the original zero lanes even after they have changed position in the matrix.

Program:

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdint.h>

#define STATE_SIZE 25
#define CAPACITY_LANES 16
#define LANE_SIZE 64

```

```

typedef struct {
    uint64_t state[STATE_SIZE];
} InternalState;

void initializeState(InternalState *state)
{
    for (int i = 0; i < STATE_SIZE; i++)
    {
        state->state[i] = 0;
    }
}

int allCapacityLanesNonzero(InternalState *state)
{
    for (int i = 0; i < CAPACITY_LANES; i++)
    {
        if (state->state[i] == 0)
        {
            return 0;
        }
    }
    return 1;
}

int main()
{
    InternalState state;
    initializeState(&state);

    srand(time(NULL));

    int steps = 0;
    while (!allCapacityLanesNonzero(&state))

```

```

{
int laneToUpdate = rand() % CAPACITY_LANES;
int bitPosition = rand() % LANE_SIZE;

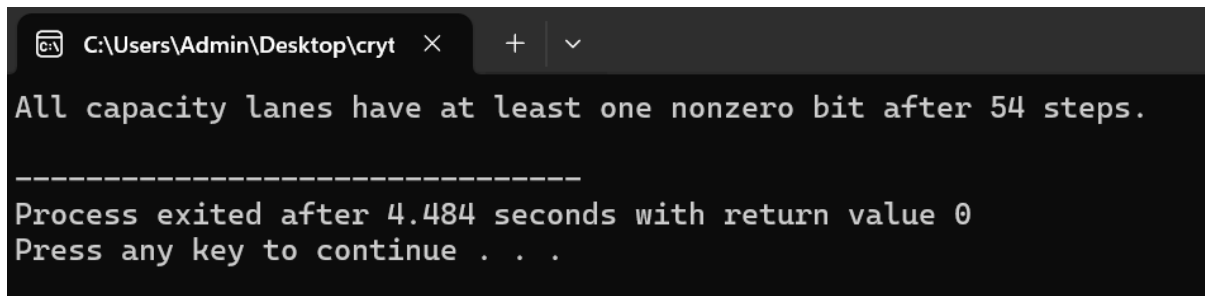
state.state[laneToUpdate] |= (1ULL << bitPosition);

steps++;
}

printf("All capacity lanes have at least one nonzero bit after %d steps.\n", steps);
return 0;
}

```

Sample output:



```

C:\Users\Admin\Desktop\crypt >
All capacity lanes have at least one nonzero bit after 54 steps.
-----
Process exited after 4.484 seconds with return value 0
Press any key to continue . . .

```

30. Write a C program for CBC MAC of a oneblock message X , say $T = \text{MAC}(K, X)$, the adversary immediately knows the CBC MAC for the two-block message $X || (X \oplus T)$ since this is once again.

Program:

```

#include <stdio.h>

#include <string.h>

void xorBlocks(const char *input1, const char *input2, char *output, int block_size) {
    for (int i = 0; i < block_size; ++i) {
        output[i] = input1[i] ^ input2[i];
    }
}

void cbcMacOneBlock(const char *key, const char *message, char *mac, int block_size)

```



```

{
    xorBlocks(key, message, mac, block_size);
}

void cbcMacTwoBlocks(const char *key, const char *message, char *mac, int block_size) {
    char t[block_size];
    cbcMacOneBlock(key, message, t, block_size);
    xorBlocks(message + block_size, t, mac, block_size);
}

int main() {
    int block_size = 8;
    char key[block_size] = "abcdefgh";
    char message[block_size] = "12345678";
    char mac[block_size];
    cbcMacOneBlock(key, message, mac, block_size);
    printf("One-block MAC: ");
    for (int i = 0; i < block_size; ++i) {
        printf("%02x", (unsigned char)mac[i]);
    }
    printf("\n");
    char messageTwoBlocks[2 * block_size] = "1234567812345678";
    char macTwoBlocks[block_size];
    cbcMacTwoBlocks(key, messageTwoBlocks, macTwoBlocks, block_size);
    printf("Two-block MAC: ");
    for (int i = 0; i < block_size; ++i) {
        printf("%02x", (unsigned char)macTwoBlocks[i]);
    }
    printf("\n");
    return 0;
}

```

Sample output

```
C:\Users\Admin\Desktop\crypt × + ▾  
One-block MAC: 5050505050505050  
Two-block MAC: 6162636465666768  
  
-----  
Process exited after 1.904 seconds with return value 0  
Press any key to continue . . . |
```