CRYPTOGRAPHY AND NETWORK SECURITY

Ayush Saini LAB - 5
23BAI1049

Title of the Experiment:

Study and Comparison of Modes of Operation using Simplified DES (S-DES)

Aim :

To implement different modes of operation using Simplified DES (S-DES), analyze their behavior on plaintext data, and compare the modes based on security and performance parameters using a real-time inspired application scenario

Objective:

1. Implement S-DES encryption for different modes of operation.

2. Observe the effect of modes on identical plaintext blocks.

3. Compare modes using suitable parameters.

4. Identify the best mode for different application requirements.

Real-Time Scenario Description:

A temperature sensor sends repeating readings every minute:

Plaintext blocks (8-bit):

P1 = 11001010

P2 = 11001010

P3 = 11001010

Key (10-bit):

K = 1010000010

Initialization Vector (IV):

IV = 11110000

System Model:

Entities Involved

- **Client System**
- **Server System**

Communication Flow

Client Side

1. Input plaintext blocks
2. Select mode (ECB / CBC / CFB / OFB / CTR)
3. Input key and IV / Counter if required
4. Encrypt blocks using S-DES
5. Send ciphertext blocks via socket

Server Side

1. Receive ciphertext blocks
2. Use same key and IV / Counter
3. Decrypt using selected mode
4. Display recovered plaintext

Network Assumption

All entities are assumed to be connected to the same Local Area Network (LAN) .

Mathematical / Cryptographic Background:

The following Cryptographic concepts form the foundation of this experiment:

Block ciphers like DES or S-DES encrypt fixed-size blocks. Modes of operation define how blocks are linked together to encrypt long messages securely.

Data Flow Description:

• User enters binary plaintext at the client system.
• Plaintext is divided into **8-bit blocks**.

- User selects the **mode of operation** (ECB, CBC, CFB, OFB, CTR).
- Client encrypts plaintext blocks using **S-DES** with the selected mode.
- IV is used for **CBC, CFB, OFB**; counter values are used for **CTR** mode.
- Encrypted ciphertext blocks are generated at the client side.
- Ciphertext and required parameters are sent to the server using **TCP sockets**.
- Server receives the encrypted data and applies the corresponding **S-DES decryption mode**.
- Original plaintext is recovered at the server.
- Attacker may capture packets but can view only encrypted data.

Client.c

```c
#include <winsock2.h>
#include <stdio.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8080
#define MAX 10

/*  FULL S-DES */

int IP[8] = {2, 6, 3, 1, 4, 8, 5, 7};
int IP_INV[8] = {4, 1, 3, 5, 7, 2, 8, 6};
int EP[8] = {4, 1, 2, 3, 2, 3, 4, 1};
int P4[4] = {2, 4, 3, 1};

int S0[4][4] = {
    {1, 0, 3, 2},
    {3, 2, 1, 0},
    {0, 2, 1, 3},
    {3, 1, 3, 2}};

int S1[4][4] = {
    {0, 1, 2, 3},
    {2, 0, 1, 3},
    {3, 0, 1, 0},
    {2, 1, 0, 3}};

void permute(int *in, int *out, int *p, int n)
{
    for (int i = 0; i < n; i++)
        out[i] = in[p[i] - 1];
}

void xor_bits(int *a, int *b, int *out, int n)
{
    for (int i = 0; i < n; i++)
        out[i] = a[i] ^ b[i];
}

void sbox(int *in, int box[4][4], int *out)
{
    int row = in[0] * 2 + in[3];
    int col = in[1] * 2 + in[2];
    int val = box[row][col];
    out[0] = (val >> 1) & 1;
    out[1] = val & 1;
}

void fk(int *bits, int *key)
{
    int L[4], R[4], ep[8], temp[8];
    int s0o[2], s1o[2], p4[4];
    for (int i = 0; i < 4; i++)
    {
        L[i] = bits[i];
        R[i] = bits[i + 4];
    }

    permute(R, ep, EP, 8);
    xor_bits(ep, key, temp, 8);

    sbox(temp, S0, s0o);
    sbox(temp + 4, S1, s1o);

    int s[4] = {s0o[0], s0o[1], s1o[0], s1o[1]};
    permute(s, p4, P4, 4);
    xor_bits(L, p4, L, 4);

    for (int i = 0; i < 4; i++)
    {
        bits[i] = L[i];
        bits[i + 4] = R[i];
    }
}

void swap(int *b)
{
    for (int i = 0; i < 4; i++)
    {
        int t = b[i];
        b[i] = b[i + 4];
        b[i + 4] = t;
    }
}

void sdes_encrypt(int *pt, int *k1, int *k2, int *ct)
{
    int t[8];
    permute(pt, t, IP, 8);
    fk(t, k1);
    swap(t);
    fk(t, k2);
    permute(t, ct, IP_INV, 8);
}

void sdes_decrypt(int *ct, int *k1, int *k2, int *pt)
{
    int t[8];
    permute(ct, t, IP, 8);
    fk(t, k2);
    swap(t);
    fk(t, k1);
    permute(t, pt, IP_INV, 8);
}

// modes of operation
void ecb_enc(int pt[][8], int ct[][8], int n, int *k1, int *k2)
{
    for (int i = 0; i < n; i++)
        sdes_encrypt(pt[i], k1, k2, ct[i]);
}

void cbc_enc(int pt[][8], int ct[][8], int n, int iv[8], int *k1, int *k2)
{
    int temp[8];
    for (int i = 0; i < n; i++)
    {
        xor_bits(pt[i], iv, temp, 8);
        sdes_encrypt(temp, k1, k2, ct[i]);
        for (int j = 0; j < 8; j++)
            iv[j] = ct[i][j];
    }
}

void cfb_enc(int pt[][8], int ct[][8], int n, int iv[8], int *k1, int *k2)
{
    int temp[8];
    for (int i = 0; i < n; i++)
    {
        sdes_encrypt(iv, k1, k2, temp);
        xor_bits(pt[i], temp, ct[i], 8);
        for (int j = 0; j < 8; j++)
            iv[j] = ct[i][j];
    }
}

void ofb_enc(int pt[][8], int ct[][8], int n, int iv[8], int *k1, int *k2)
{
    int stream[8];
    for (int i = 0; i < n; i++)
    {
        sdes_encrypt(iv, k1, k2, stream);
        xor_bits(pt[i], stream, ct[i], 8);
        for (int j = 0; j < 8; j++)
            iv[j] = stream[j];
    }
}

void ctr_enc(int pt[][8], int ct[][8], int n, int ctr[][8], int *k1, int *k2)
{
    int stream[8];
    for (int i = 0; i < n; i++)
    {
        sdes_encrypt(ctr[i], k1, k2, stream);
        xor_bits(pt[i], stream, ct[i], 8);
    }
}

int main()
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;

    int pt[MAX][8], ct[MAX][8], ctr[MAX][8];
    int iv[8], k1[8], k2[8];
    int blocks, mode;

    WSAStartup(MAKEWORD(2, 2), &wsa);
    s = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    server.sin_addr.s_addr = inet_addr("127.0.0.1");

    connect(s, (struct sockaddr *)&server, sizeof(server));

    printf("Blocks: ");
    scanf("%d", &blocks);
    printf("Plaintext:\n");
    for (int i = 0; i < blocks; i++)
        for (int j = 0; j < 8; j++)
            scanf("%d", &pt[i][j]);

    printf("Mode 1-ECB 2-CBC 3-CFB 4-OFB 5-CTR: ");
    scanf("%d", &mode);

    printf("Enter K1 (8): ");
    for (int i = 0; i < 8; i++)
        scanf("%d", &k1[i]);
    printf("Enter K2 (8): ");
    for (int i = 0; i < 8; i++)
        scanf("%d", &k2[i]);

    if (mode != 1 && mode != 5)
    {
        printf("Enter IV: ");
        for (int i = 0; i < 8; i++)
            scanf("%d", &iv[i]);
    }
    if (mode == 5)
    {
        printf("Enter Counter blocks:\n");
        for (int i = 0; i < blocks; i++)
            for (int j = 0; j < 8; j++)
                scanf("%d", &ctr[i][j]);
    }

    if (mode == 1)
        ecb_enc(pt, ct, blocks, k1, k2);
    if (mode == 2)
        cbc_enc(pt, ct, blocks, iv, k1, k2);
    if (mode == 3)
        cfb_enc(pt, ct, blocks, iv, k1, k2);
    if (mode == 4)
        ofb_enc(pt, ct, blocks, iv, k1, k2);
    if (mode == 5)
        ctr_enc(pt, ct, blocks, ctr, k1, k2);

    send(s, (char *)&mode, sizeof(mode), 0);
    send(s, (char *)&blocks, sizeof(blocks), 0);
    if (mode != 1 && mode != 5)
        send(s, (char *)iv, sizeof(iv), 0);
    if (mode == 5)
        send(s, (char *)ctr, sizeof(ctr), 0);
    send(s, (char *)ct, sizeof(ct), 0);

    printf("Ciphertext sent\n");

    closesocket(s);
    WSACleanup();
    return 0;
}
```

Server.c

```
#include <winsock2.h>
#include <stdio.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8080
#define MAX 10
```

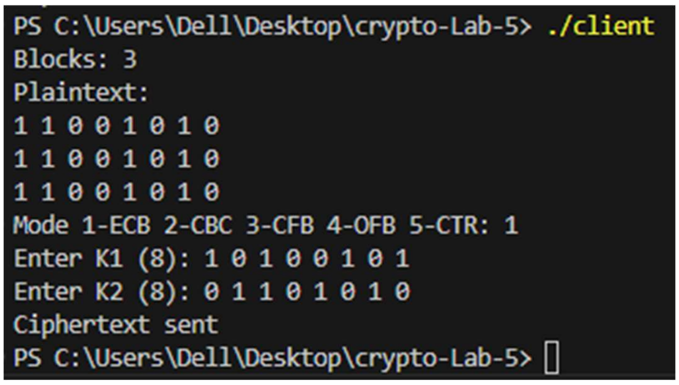
```c
/*  FULL S-DES */

int IP[8] = {2, 6, 3, 1, 4, 8, 5, 7};
int IP_INV[8] = {4, 1, 3, 5, 7, 2, 8, 6};
int EP[8] = {4, 1, 2, 3, 2, 3, 4, 1};
int P4[4] = {2, 4, 3, 1};

int S0[4][4] = {
    {1, 0, 3, 2},
    {3, 2, 1, 0},
    {0, 2, 1, 3},
    {3, 1, 3, 2}};

int S1[4][4] = {
    {0, 1, 2, 3},
    {2, 0, 1, 3},
    {3, 0, 1, 0},
    {2, 1, 0, 3}};

void permute(int *in, int *out, int *p, int n)
{
    for (int i = 0; i < n; i++)
        out[i] = in[p[i] - 1];
}

void xor_bits(int *a, int *b, int *out, int n)
{
    for (int i = 0; i < n; i++)
        out[i] = a[i] ^ b[i];
}

void sbox(int *in, int box[4][4], int *out)
{
    int row = in[0] * 2 + in[3];
    int col = in[1] * 2 + in[2];
    int val = box[row][col];
    out[0] = (val >> 1) & 1;
    out[1] = val & 1;
}

void fk(int *bits, int *key)
{
    int L[4], R[4], ep[8], temp[8];
    int s0o[2], s1o[2], p4[4];

    for (int i = 0; i < 4; i++)
    {
        L[i] = bits[i];
        R[i] = bits[i + 4];
    }

    permute(R, ep, EP, 8);
    xor_bits(ep, key, temp, 8);

    sbox(temp, S0, s0o);
    sbox(temp + 4, S1, s1o);

    int s[4] = {s0o[0], s0o[1], s1o[0], s1o[1]};
    permute(s, p4, P4, 4);
    xor_bits(L, p4, L, 4);

    for (int i = 0; i < 4; i++)
    {
        bits[i] = L[i];
        bits[i + 4] = R[i];
    }
}

void swap(int *b)
{
    for (int i = 0; i < 4; i++)
    {
        int t = b[i];
        b[i] = b[i + 4];
        b[i + 4] = t;
    }
}

void sdes_encrypt(int *pt, int *k1, int *k2, int *ct)
{
    int t[8];
    permute(pt, t, IP, 8);
    fk(t, k1);
    swap(t);
    fk(t, k2);
    permute(t, ct, IP_INV, 8);
}

void sdes_decrypt(int *ct, int *k1, int *k2, int *pt)
{
    int t[8];
    permute(ct, t, IP, 8);
    fk(t, k2);
    swap(t);
    fk(t, k1);
    permute(t, pt, IP_INV, 8);
}

// modes of operation

void ecb_dec(int ct[][8], int pt[][8], int n, int *k1, int *k2)
{
    for (int i = 0; i < n; i++)
        sdes_decrypt(ct[i], k1, k2, pt[i]);
}

void cbc_dec(int ct[][8], int pt[][8], int n, int iv[8], int *k1, int *k2)
{
    int temp[8], prev[8];
    for (int i = 0; i < 8; i++)
        prev[i] = iv[i];
    for (int i = 0; i < n; i++)
    {
        sdes_decrypt(ct[i], k1, k2, temp);
        xor_bits(temp, prev, pt[i], 8);
        for (int j = 0; j < 8; j++)
            prev[j] = ct[i][j];
    }
}

void cfb_dec(int ct[][8], int pt[][8], int n, int iv[8], int *k1, int *k2)
{
    int temp[8];
    for (int i = 0; i < n; i++)
    {
        sdes_encrypt(iv, k1, k2, temp);
        xor_bits(ct[i], temp, pt[i], 8);
        for (int j = 0; j < 8; j++)
```

```
            iv[j] = ct[i][j];
    }
}

void ofb_dec(int ct[][8], int pt[][8], int n, int iv[8], int *k1, int *k2)
{
    int stream[8];
    for (int i = 0; i < n; i++)
    {
        sdes_encrypt(iv, k1, k2, stream);
        xor_bits(ct[i], stream, pt[i], 8);
        for (int j = 0; j < 8; j++)
            iv[j] = stream[j];
    }
}

void ctr_dec(int ct[][8], int pt[][8], int n, int ctr[][8], int *k1, int *k2)
{
    int stream[8];
    for (int i = 0; i < n; i++)
    {
        sdes_encrypt(ctr[i], k1, k2, stream);
        xor_bits(ct[i], stream, pt[i], 8);
    }
}

int main()
{
    WSADATA wsa;
    SOCKET s, cs;
    struct sockaddr_in server, client;
    int len = sizeof(client);

    int ct[MAX][8], pt[MAX][8], ctr[MAX][8];
    int iv[8], k1[8], k2[8];
    int blocks, mode;

    WSAStartup(MAKEWORD(2, 2), &wsa);
    s = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);

    bind(s, (struct sockaddr *)&server, sizeof(server));
    listen(s, 3);

    printf("Server waiting...\n");
    cs = accept(s, (struct sockaddr *)&client, &len);

    recv(cs, (char *)&mode, sizeof(mode), 0);
    recv(cs, (char *)&blocks, sizeof(blocks), 0);
    if (mode != 1 && mode != 5)
        recv(cs, (char *)iv, sizeof(iv), 0);
    if (mode == 5)
        recv(cs, (char *)ctr, sizeof(ctr), 0);
    recv(cs, (char *)ct, sizeof(ct), 0);

    printf("\nReceived Ciphertext at Server:\n");
    for (int i = 0; i < blocks; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            printf("%d", ct[i][j]);
```

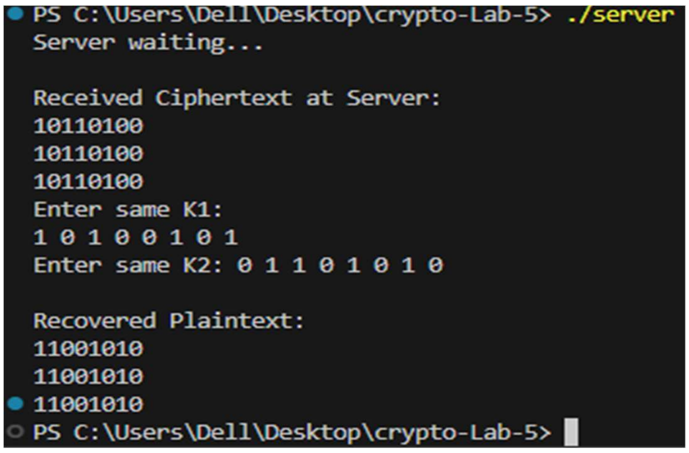
```c
        }
        printf("\n");
    }

    printf("Enter same K1: ");
    for (int i = 0; i < 8; i++)
        scanf("%d", &k1[i]);
    printf("Enter same K2: ");
    for (int i = 0; i < 8; i++)
        scanf("%d", &k2[i]);

    if (mode == 1)
        ecb_dec(ct, pt, blocks, k1, k2);
    if (mode == 2)
        cbc_dec(ct, pt, blocks, iv, k1, k2);
    if (mode == 3)
        cfb_dec(ct, pt, blocks, iv, k1, k2);
    if (mode == 4)
        ofb_dec(ct, pt, blocks, iv, k1, k2);
    if (mode == 5)
        ctr_dec(ct, pt, blocks, ctr, k1, k2);

    printf("\nRecovered Plaintext:\n");
    for (int i = 0; i < blocks; i++)
    {
        for (int j = 0; j < 8; j++)
            printf("%d", pt[i][j]);
        printf("\n");
    }

    closesocket(cs);
    closesocket(s);
    WSACleanup();
    return 0;
}
```

OUTPUTS

1.ECB

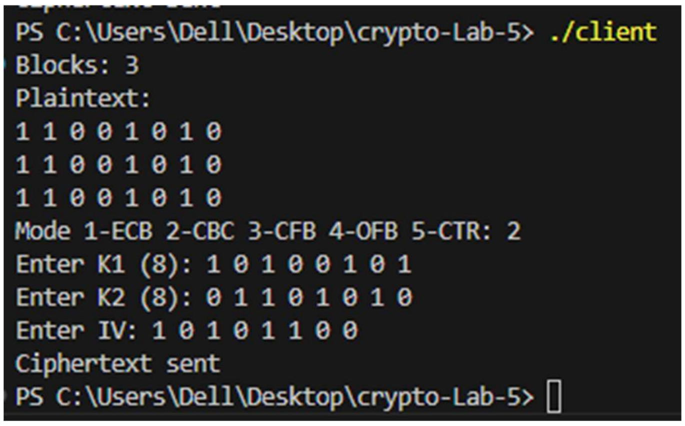


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./client
Blocks: 3
Plaintext:
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
Mode 1-ECB 2-CBC 3-CFB 4-OFB 5-CTR: 1
Enter K1 (8): 1 0 1 0 0 1 0 1
Enter K2 (8): 0 1 1 0 1 0 1 0
Ciphertext sent
PS C:\Users\Dell\Desktop\crypto-Lab-5> 
```

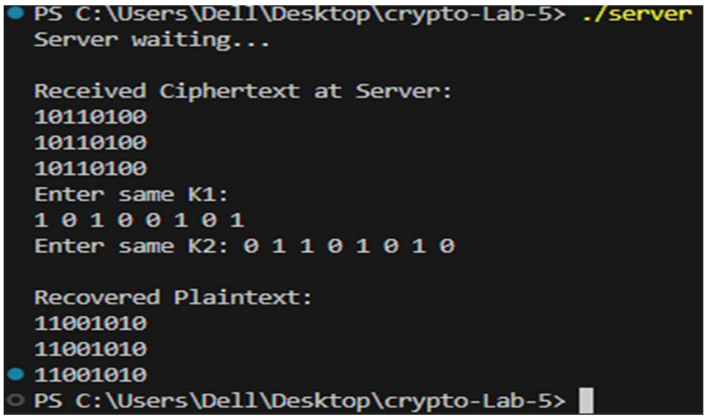


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./server
Server waiting...

Received Ciphertext at Server:
10110100
10110100
10110100
Enter same K1:
1 0 1 0 0 1 0 1
Enter same K2: 0 1 1 0 1 0 1 0

Recovered Plaintext:
11001010
11001010
11001010
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

2.cbc

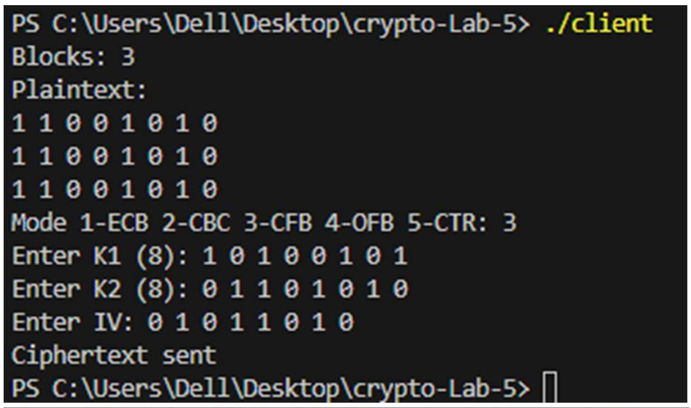


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./client
Blocks: 3
Plaintext:
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
Mode 1-ECB 2-CBC 3-CFB 4-OFB 5-CTR: 2
Enter K1 (8): 1 0 1 0 0 1 0 1
Enter K2 (8): 0 1 1 0 1 0 1 0
Enter IV: 1 0 1 0 1 1 0 0
Ciphertext sent
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

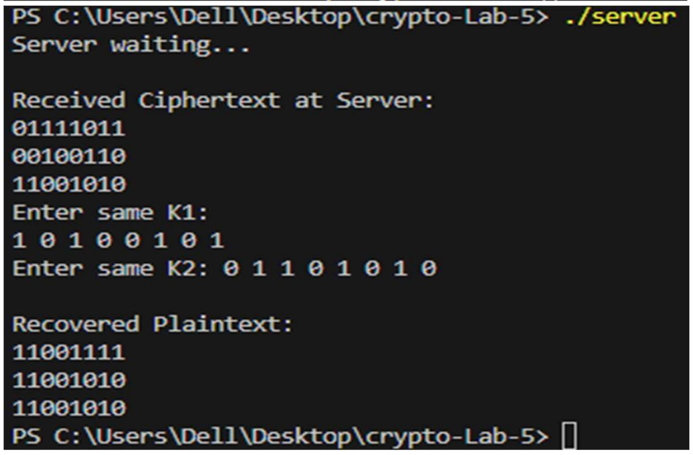


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./server
Server waiting...

Received Ciphertext at Server:
10110100
10110100
10110100
Enter same K1:
1 0 1 0 0 1 0 1
Enter same K2: 0 1 1 0 1 0 1 0

Recovered Plaintext:
11001010
11001010
11001010
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

3.CFB

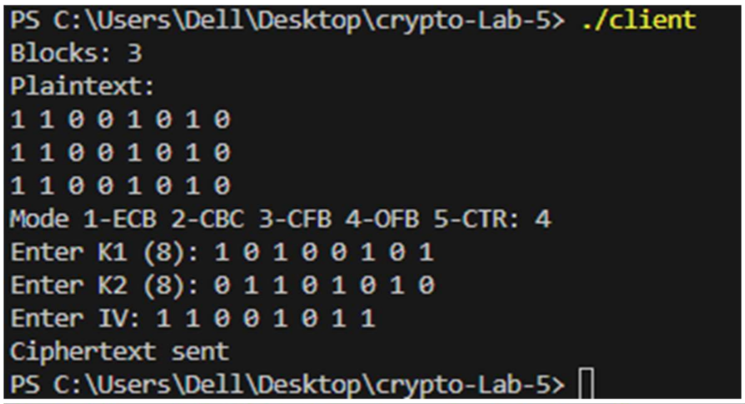


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./client
Blocks: 3
Plaintext:
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
Mode 1-ECB 2-CBC 3-CFB 4-OFB 5-CTR: 3
Enter K1 (8): 1 0 1 0 0 1 0 1
Enter K2 (8): 0 1 1 0 1 0 1 0
Enter IV: 0 1 0 1 1 0 1 0
Ciphertext sent
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

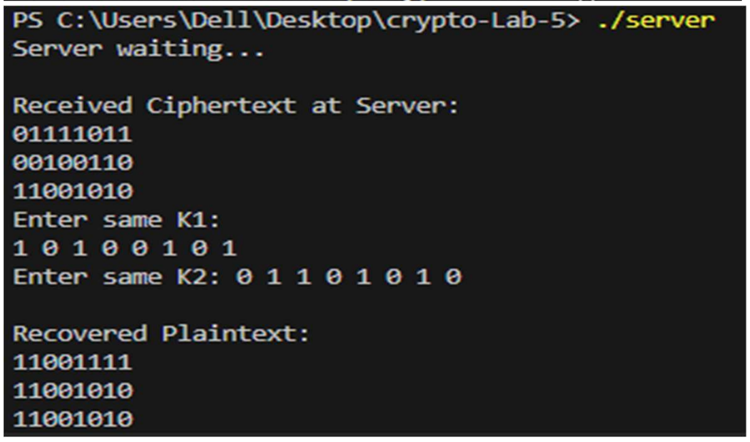


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./server
Server waiting...

Received Ciphertext at Server:
01111011
00100110
11001010
Enter same K1:
1 0 1 0 0 1 0 1
Enter same K2: 0 1 1 0 1 0 1 0

Recovered Plaintext:
11001111
11001010
11001010
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

4.OFB

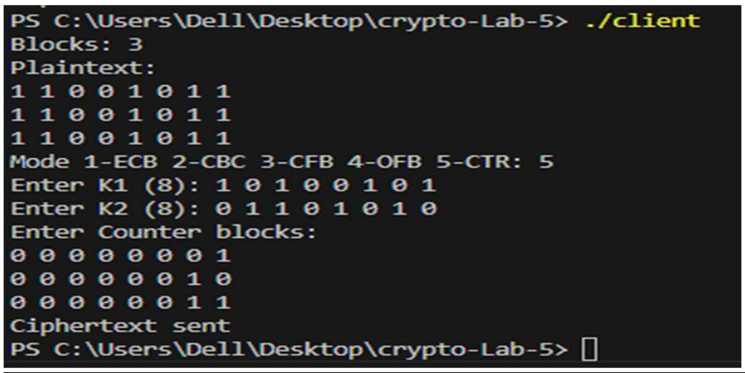


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./client
Blocks: 3
Plaintext:
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
Mode 1-ECB 2-CBC 3-CFB 4-OFB 5-CTR: 4
Enter K1 (8): 1 0 1 0 0 1 0 1
Enter K2 (8): 0 1 1 0 1 0 1 0
Enter IV: 1 1 0 0 1 0 1 1
Ciphertext sent
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

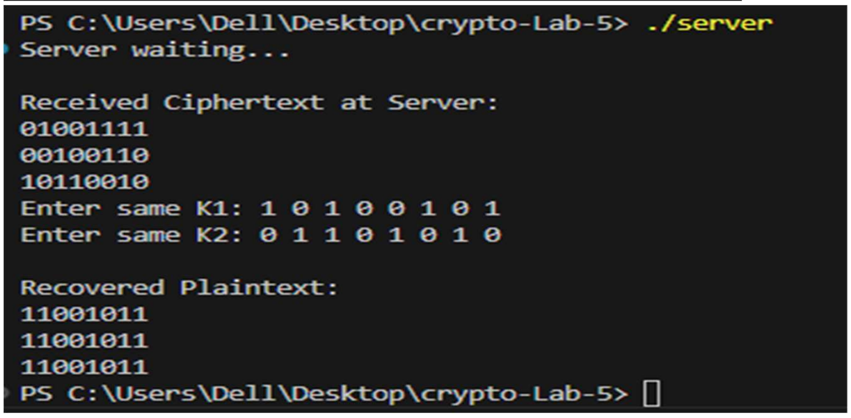


```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./server
Server waiting...

Received Ciphertext at Server:
01111011
00100110
11001010
Enter same K1:
1 0 1 0 0 1 0 1
Enter same K2: 0 1 1 0 1 0 1 0

Recovered Plaintext:
11001111
11001010
11001010
```

5.CTR



```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./client
Blocks: 3
Plaintext:
1 1 0 0 1 0 1 1
1 1 0 0 1 0 1 1
1 1 0 0 1 0 1 1
Mode 1-ECB 2-CBC 3-CFB 4-OFB 5-CTR: 5
Enter K1 (8): 1 0 1 0 0 1 0 1
Enter K2 (8): 0 1 1 0 1 0 1 0
Enter Counter blocks:
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1
Ciphertext sent
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```



```
PS C:\Users\Dell\Desktop\crypto-Lab-5> ./server
Server waiting...

Received Ciphertext at Server:
01001111
00100110
10110010
Enter same K1: 1 0 1 0 0 1 0 1
Enter same K2: 0 1 1 0 1 0 1 0

Recovered Plaintext:
11001011
11001011
11001011
PS C:\Users\Dell\Desktop\crypto-Lab-5>
```

Security Goal:

- To ensure **confidentiality** of data during transmission
- To prevent unauthorized access to plaintext information
- To protect data from network eavesdropping attacks
- To demonstrate secure client–server communication using encryption
- To show the importance of secret key protection

Prevention Mechanism and Security Reinforcement:

- Data is encrypted using **Simplified DES (S-DES)** before transmission.
- A **shared symmetric secret key** is used by both client and server.
- Encrypted **ciphertext** is transmitted instead of plaintext over the network.
- Packet capture using **Wireshark reveals only encrypted binary data** to attackers.
- Brute-force attack is limited and performed only for **demonstration purposes**.
- The experiment highlights the weakness of S-DES due to its small key size.
- Stronger encryption algorithms such as **AES** are recommended for real-world use.
- Secure key management and network monitoring practices are essential to enhance security.

Implementation Details

● Programming Language Used: C

● Core Functions Implemented:
• Encryption Algorithm: Simplified DES (S-DES)
• Key Type: Symmetric secret key (10-bit)
• Modes of Operation: CBC, CFB, OFB, CTR
• Initialization Parameters:
– Initialization Vector (IV) for CBC, CFB, OFB
– Counter values for CTR mode
• Communication Method: TCP socket programming (Winsock on Windows)
• Data Transfer: Encrypted binary ciphertext over the network

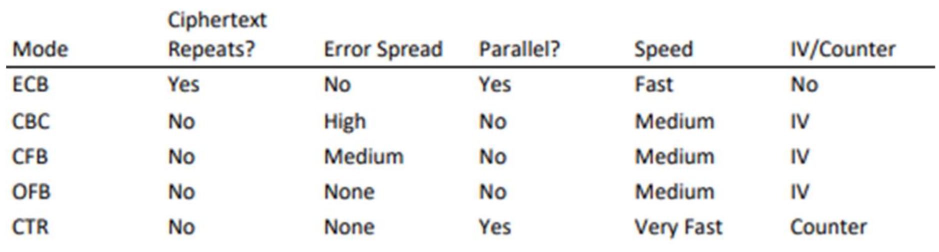
- **Input** : Plaintext message,key entered by the user
- **Output :**
    - Encrypted ciphertext generated at the client
    - Ciphertext transmitted over the network
    - Decrypted plaintext recovered at the server

Results and Analysis:

S-DES encryption using **CBC, CFB, OFB, and CTR modes** was successfully implemented in a client–server environment.
The server correctly decrypted the received ciphertext, and Wireshark captured only encrypted binary data.

The experiment shows that modes like **CBC, CFB, OFB, and CTR** improve security by hiding plaintext patterns compared to ECB.
Although suitable for demonstration, **S-DES is vulnerable to brute-force attacks**, so stronger algorithms like **AES** are recommended.

Result Table



Mode	Ciphertext Repeats?	Error Spread	Parallel?	Speed	IV/Counter
ECB	Yes	No	Yes	Fast	No
CBC	No	High	No	Medium	IV
CFB	No	Medium	No	Medium	IV
OFB	No	None	No	Medium	IV
CTR	No	None	Yes	Very Fast	Counter

References:

- William Stallings – *Cryptography and Network Security*
- NPTEL – Cryptography and Network Security Lectures
- RFC 4107 – Cryptographic Algorithm Implementation Requirement.