#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

#define DECRYPT\_KEY 4 // Same random value to subtract from ASCII

void decrypt\_message(char \*message) {

for (int i = 0; message[i] != '\0'; i++) {

message[i] -= DECRYPT\_KEY;

}

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Bind the socket to the network

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Read the encrypted message

read(new\_socket, buffer, BUFFER\_SIZE);

printf("Encrypted message received: %s\n", buffer);

// Decrypt the message

decrypt\_message(buffer);

printf("Decrypted message: %s\n", buffer);

// Close the connection

close(new\_socket);

close(server\_fd);

return 0;

}

C CODE #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

|  |  |
| --- | --- |
|  |  |
|  |  |

#define PORT 8080

#define BUFFER\_SIZE 1024

#define ENCRYPT\_KEY 4 // Random value to add to ASCII

void encrypt\_message(char \*message) {

for (int i = 0; message[i] != '\0'; i++) {

message[i] += ENCRYPT\_KEY;

}

}

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char message[BUFFER\_SIZE];

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

// Input message

printf("Enter a message: ");

fgets(message, BUFFER\_SIZE, stdin);

message[strcspn(message, "\n")] = 0; // Remove trailing newline

// Encrypt message

encrypt\_message(message);

// Send encrypted message to server

send(sock, message, strlen(message), 0);

printf("Encrypted message sent to server: %s\n", message);

// Close the socket

close(sock);

return 0;

}