# Experiment 8: Alphanumeric Cipher using TCP Sockets

**Aim:** To use TCP Sockets to implement simple alphanumeric ciphers

**Objective:** After carrying out this experiment, students will be able to:

* Apply simple alphanumeric cipher techniques to encrypt and decrypt text messages

**Problem statement:** You are required to write programs for a TCP client-server system that performs simple encryption and decryption using alphanumeric ciphers. The client should first authenticate a user using a pre-defined password. Then the client (user) sends a message to the server. The server will encrypt the message using alphanumeric cipher and store it in a file. Subsequently, when a user asks for the message he has entered previously, the message should be decrypted and transmitted back to the client.

**Analysis:** While analyzing your program, you are required to address the following points:

* From the security perspective, how can this implementation be made more secure?
* What are the vulnerabilities of alphanumeric ciphers?

**MARKS DISTRIBUTION**

|  |  |  |
| --- | --- | --- |
| **Component** | **Maximum Marks** | **Marks Obtained** |
| Preparation of Document | 7 |  |
| Results | 7 |  |
| Viva | 6 |  |
| **Total** | **20** |  |

Submitted by:

Register No:

1. Algorithm/Flowchart
2. Program

CLIENT CODE:

// Client side C/C++ program to demonstrate Socket programming

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

#include <iostream>

bool authenticated() {

    using namespace std;

    std::string pass("meow123");

    cout << "Enter the auth pass : ";

    std::string input;

    cin >> input;

    return pass == input;

}

int main(int argc, char const \*argv[]) {

    if (not authenticated()) {

        std::cout << "AUTH ERROR !" << std::endl;

        exit(EXIT\_FAILURE);

    }

    int sockfd = 0, valread;

    struct sockaddr\_in serv\_addr;

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

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

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : SOCKET CREATED " << std::endl;

    }

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

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

    // Convert IPv4 and IPv6 addresses 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;

    } else {

        std::cout << argv[0] << " : ADDRESS CONVERTED" << std::endl;

    }

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

        printf("\nConnection Failed \n");

        return -1;

    } else {

        std::cout << argv[0] << " : CONNECTION ESTABLISHED" << std::endl;

    }

    char buffer[1024] = {0};

    while(true) {

        std::cout << "1.STORE\t2.FETCH\t3.EXIT\nCHOICE : ";

        int choice;

        std::cin >> choice;

        switch(choice) {

            case 1: {

                std::cout << "Enter the message to send : " << std::endl;

                std::string input;

                std::cin.ignore();

                std::getline(std::cin, input);

                send(sockfd, "STO", sizeof("STO"), 0);

                send(sockfd, input.c\_str(), input.size(), 0);

                std::cout << argv[0] << " : INPUT MESSAGE SENT !" << std::endl;

                bzero(buffer, sizeof(buffer));

                recv(sockfd, buffer, sizeof(buffer), 0);

                std::cout << argv[0] << " : RECIEVED FROM SERVER : " << buffer << std::endl;

            } break;

            case 2: {

                send(sockfd, "FET", sizeof("FET"), 0);

                std::cout << argv[0] << " : FETCH INSTRUCTION SENT !" << std::endl;

                bzero(buffer, sizeof(buffer));

                recv(sockfd, buffer, sizeof(buffer), 0);

                std::cout << argv[0] << " : RECIEVED FROM SERVER : " << buffer << std::endl;

            } break;

            default: {

                send(sockfd, "EXIT", sizeof("EXIT"), 0);

                close(sockfd);

                exit(EXIT\_SUCCESS);

            }

        }

    }

    return 0;

}

SERVER CODE:

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

unsigned long long pow36[11];

void calc\_pow\_36();

unsigned long long base36encode(std::string);

std::string base36decode(unsigned long long message);

char base36chars[37] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main(int argc, char const \*argv[]) {

    calc\_pow\_36();

    int server\_fd, new\_socket, valread;

    struct sockaddr\_in address;

    int opt = 1;

    int addrlen = sizeof(address);

    // Creating socket file descriptor

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

        perror("socket failed");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : " << "SOCKET CREATED SUCCESSFULLY" << std::endl;

    }

    // Forcefully attaching socket to the port 8080

    if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt, sizeof(opt))) {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    address.sin\_family = AF\_INET;

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

    address.sin\_port = htons( PORT );

    // Forcefully attaching socket to the port 8080

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

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }  else {

        std::cout << argv[0] << " : " << "BIND SUCCESSFUL TO PORT " << PORT << std::endl;

    }

    if (listen(server\_fd, 3) < 0) {

        perror("listen");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : " << "NOW LISTENING" << std::endl;

    }

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

        perror("accept");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : CLIENT ACCEPTED" << std::endl;

    }

    char buffer[1024] = {0};

    // fetch the operation from the client

    while (true) {

        bzero(buffer, 1024);

        valread = read( new\_socket , buffer, sizeof(buffer));

        std::string operation(buffer);

        std::cout << argv[0] << " : OPERATION RECIEVED : " << buffer << std::endl;

        if (operation == "STO") {

            std::ofstream outfile;

            outfile.open("STORE.dat", std::ios::out);

            bzero(buffer, 1024);

            read(new\_socket, buffer, sizeof(buffer));

            std::string recieved(buffer);

            unsigned long long encoded = base36encode(recieved);

            if (encoded == 0) {

                write(new\_socket, "FAILED", sizeof("FAILED"));

            } else {

                write(new\_socket, "SUCCESS", sizeof("SUCCESS"));

                outfile << encoded;

                std::cout << argv[0] << " : ENCODED : " << encoded << std::endl;

            }

            outfile.close();

        } else if (operation == "FET") {

            std::ifstream infile;

            infile.open("STORE.dat", std::ios::in);

            std::string val;

            infile >> val;

            std::string decoded\_val = base36decode(std::stoull(val));

            write(new\_socket, decoded\_val.c\_str(), decoded\_val.size());

            infile.close();

        } else {

            break;

        }

    }

    // Close the Server File Descriptor

    close(server\_fd);

    return 0;

}

void calc\_pow\_36() {

    pow36[0] = 1;

    for (int i = 0 ; i <= 9 ; i++) {

        pow36[i+1] = pow36[i] \* 36ull;

    }

}

unsigned long long base36encode(std::string message) {

    unsigned long long n;

    unsigned long long encoded = 0;

    std::reverse(message.begin(), message.end());

    int i = 0;

    for (char& c : message) {

        if (c >= '0' and c <= '9') {

            n = c - '0';

        } else if (c >= 'A' and c <= 'Z') {

            n = c - 'A' + 10;

        } else if (c >= 'a' and c <= 'z') {

            n = c - 'a' + 10;

        } else {

            return 0;

        }

        encoded += pow36[i] \* n;

        i++;

    }

    return encoded;

}

std::string base36decode(unsigned long long message) {

    std::string decoded("");

    while(message != 0) {

        decoded.push\_back(base36chars[message % 36]);

        message /= 36;

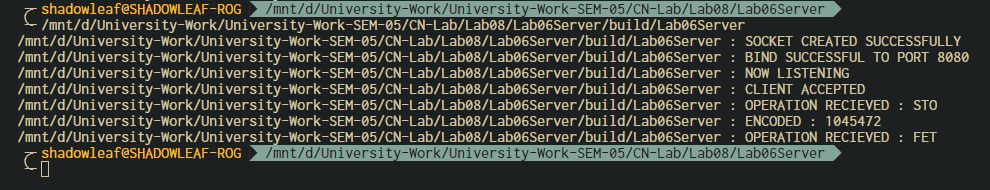
    }

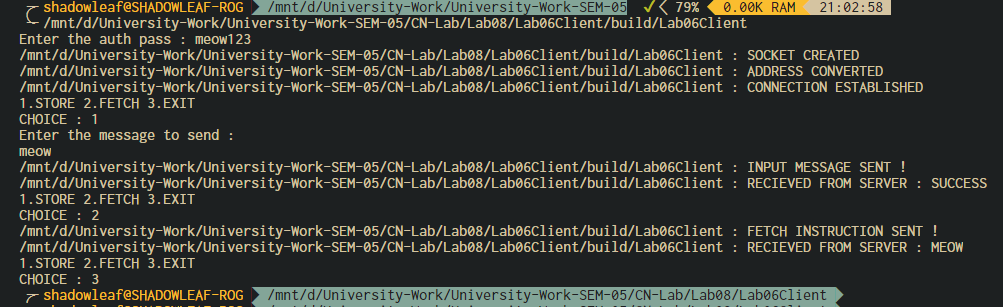
    std::reverse(decoded.begin(), decoded.end());

    return decoded;

}

1. Results





1. Analysis and Discussions
2. Conclusions
3. Comments
   1. Limitations of the experiment
   2. Limitations of the results obtained
   3. Learning
   4. Recommendations