

**Cómputo Distribuido**

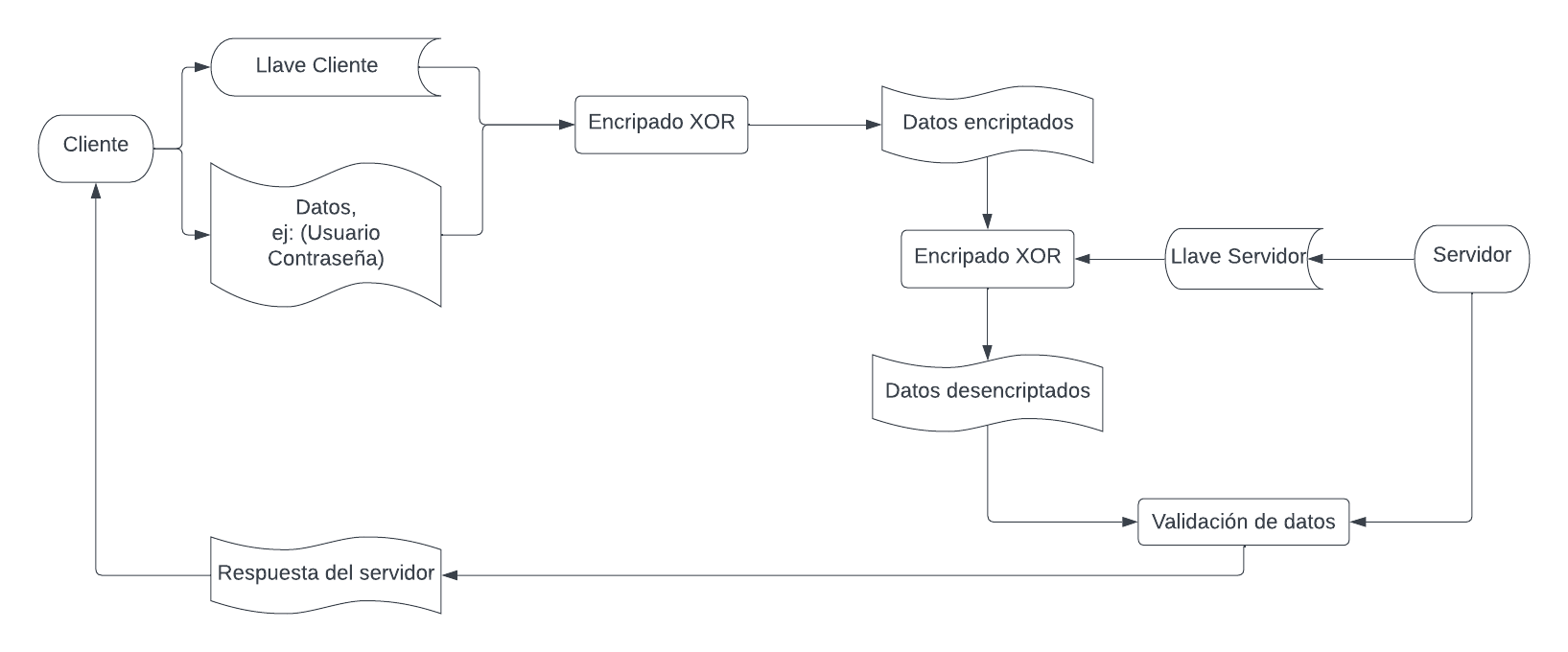
Profesor: Dr. Juan Carlos López Pimentel

**PROYECTO FINAL**

Carlos Manuel Vélez

Andrés Martínez Cabrera

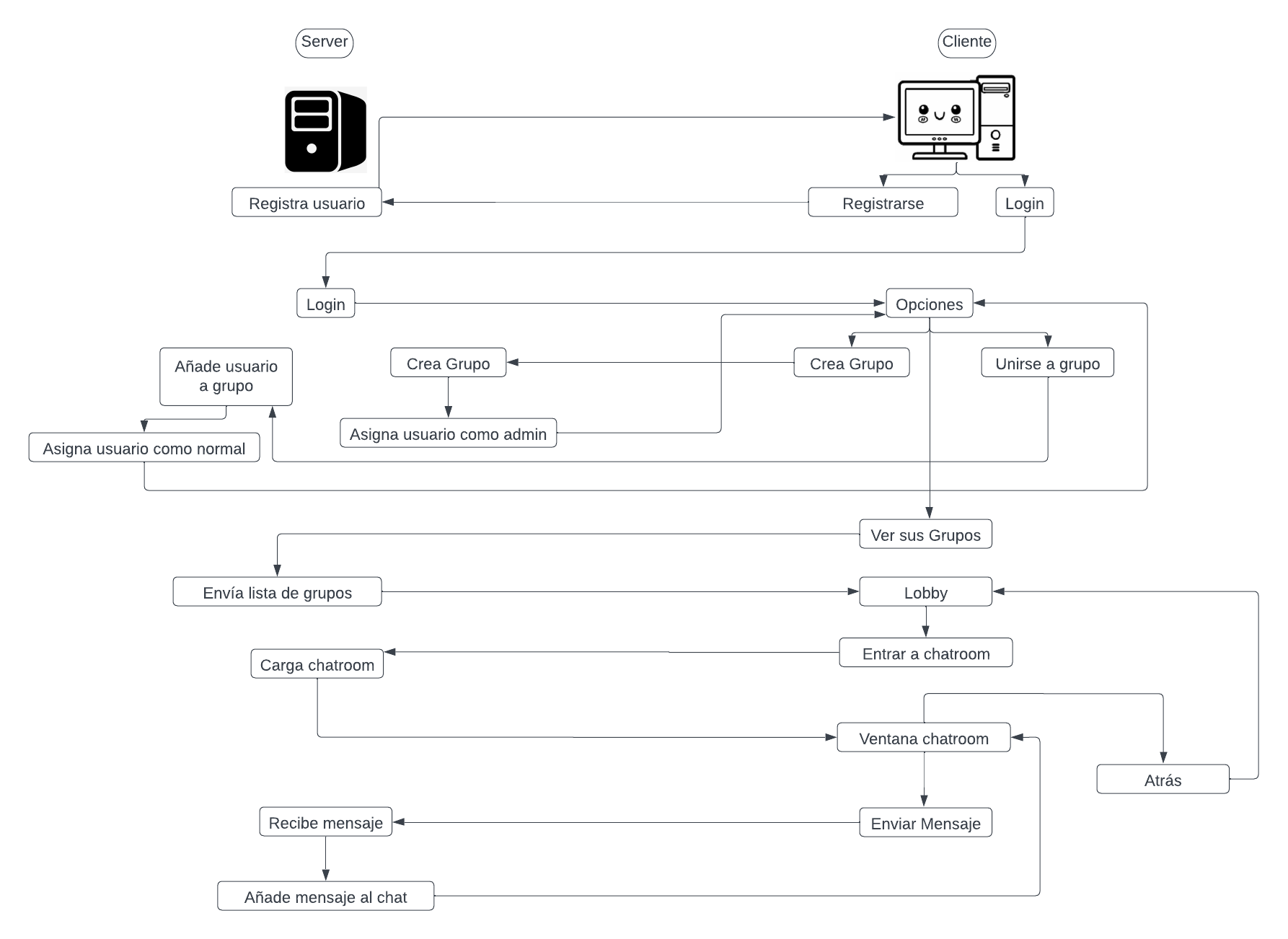
Miguel Angel Tovar Rodríguez

**Especificación Alice and Bob de protocolos diseñados**

**Explicación de los protocolos**

Usamos una autentificación cifrada, la cual encripta usuario y contraseña, los manda al servidor, él los desencripta y valida si son correctos.

Para la comunicación usamos protocolo TCP

**Diagrama que ilustre la arquitectura con las tecnologías**

**Explicación del diagrama**

Empezando tenemos la opción de ingresar o registrarse, dependiendo de la opción elegida, pedirá datos y los enviará al servidor, el cual regresara mensaje de aceptación.

Si se logra ingresar, manda a una pantalla de opciones, en donde podrás crear un grupo, unirte a un grupo o ver tus grupos.

Si eliges crear un grupo o unirte a un grupo, te pedirá el nombre del grupo y se creará o te agregará, cual sea el caso. Si se crea un grupo, te asigna rol de admin, si solo te añade a un grupo se asigna rol de normal.

En la opción de ver tus grupos, te mandará a un lobby donde verás todos tus grupos y podrás ingresar a ellos.

Cuando ingresas a uno de tus grupos verás el historial de mensajes en el chatroom y podrás enviar mensajes nuevos.

De la pantalla de chatroom puedes regresar al lobby y entrar a otro grupo.

En cualquier momento se puede cerrar la ventana y esto detendrá el programa.

**Conclusiones**

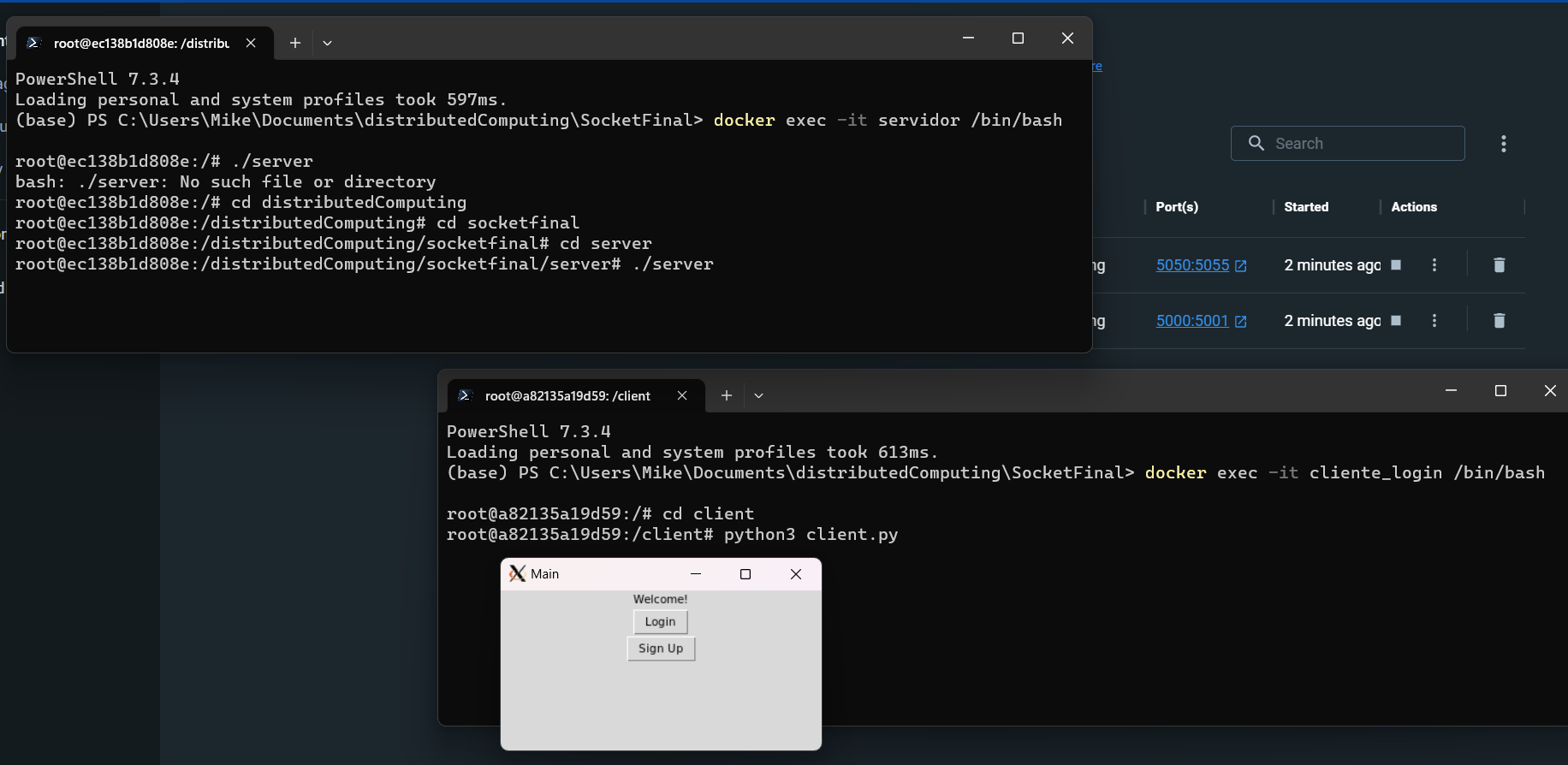
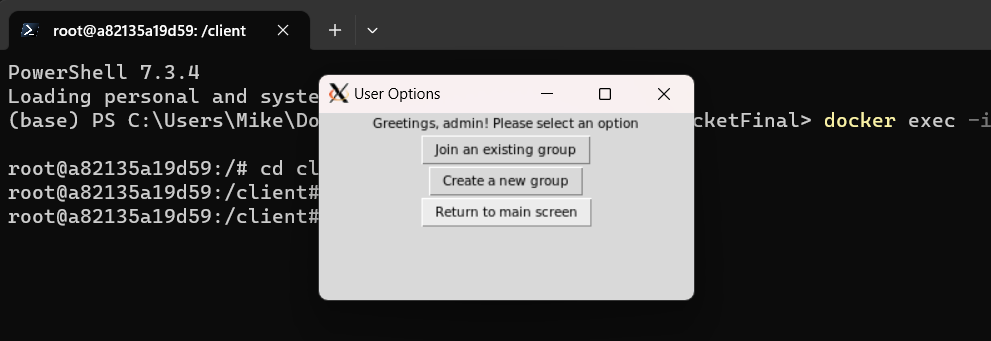
El proyecto no logramos completarlo al 100% como hubiéramos deseado. Tuvimos algunos problemas de manejo de caracteres, ya que al momento de pasar de socket a algún lenguaje, se agregaban caracteres que no deseábamos.

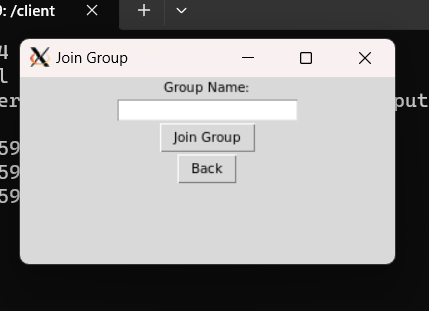
Las partes en las que tuvimos problemas fue al mostrar los grupos a los que pertenece un usuario **esto se tiene que modificar cuando terminemosssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss**

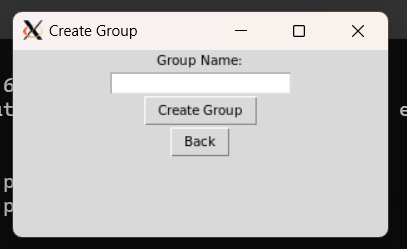
Lo que más se nos dificultó, y con lo que insistimos, es el manejo de un lenguaje con una sintaxis tan poco amigable con el usuario, siendo usuarios con nula experiencia. Al no tener experiencia manejando datos en lenguaje c, tuvimos muchos errores, investigando resolvimos todos los que pudimos, pero consideramos que el buen uso de ese lenguaje requiere mucha práctica, cosa que se nos dificultó ya que todos los miembros del equipo trabajamos y tenemos otros proyectos. El uso de herramientas que no ofrecen las soluciones más prácticas es un lastre con el que venimos cargando desde el principio de semestre, nos hubiera ahorrado horas de trabajo el poder elegir en qué lenguaje se puede desarrollar el servidor, **ya que el resultado sería el mismo idealmente si se usara un lenguaje u otro**, simplemente al elegir el lenguaje en el que vas a desarrollar puedes hacer uso de las herramientas que se ofrecen y que ya estás acostumbrado a usar, lo cual disminuye considerablemente la aparición de errores y las horas de codificación.

En cuanto a lo aprendido, logramos el objetivo de aprender cómo funcionan los diversos protocolos, soluciones a problemas comunes como la disponibilidad y la concurrencia, quedaron claros y estamos completamente agradecidos y conformes con esta parte, así como agradecemos la experiencia del profesor al haber aplicado estas herramientas para la resolución de problemas reales en el campo laboral, nos transmitió muchos consejos y experiencia a lo largo del semestre.

Aprendimos también a usar el lenguaje c, aunque fue solo un poco ya que para aprenderlo bien necesitamos dedicarle bastante tiempo, el cual no tenemos y ninguno de nosotros pretende usarlo en la vida laboral, por lo que tampoco tenemos el interés de cumplir una exigencia que se basa en una preferencia personal de alguien más con el argumento de que “así es como se hacía”, sabemos que algunos lenguajes tienen ventajas sobre otros, como manejo de memoria y rapidez de ejecución, pero actualmente lo que más nos interesa como estudiantes es optimizar tiempo a la hora de codificar soluciones, ya que como se mencionó antes, contamos con otras responsabilidades y no nos parece justo tener que adaptarnos a las preferencias de alguien más cuando no es completamente necesario y da la sensación de que se toman decisiones con la finalidad de dificultar a los estudiantes sus labores.







**CLIENT:**

import tkinter as tk

from tkinter import messagebox

import socket

def send\_to\_server(data, operation):

    if data == False: return

     # Crea un socket TCP/IP

    client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Conecta el cliente al servidor en la direccion y puerto especificados

    server\_address = ('172.18.2.3', 5000)

    client\_socket.connect(server\_address)

    username = data[0]

    if operation == "Login": operation = 1

    elif operation == "Sign Up": operation = 2

    elif operation == "Join Group": operation = 3

    elif operation == "Create Group": operation = 4

    try:

        # Envía los valores de inicio de sesion al servidor

        # data[1] puede ser password o groupname, dependiendo de la opcion seleccionada

        message = f"{username}:{data[1]}:{operation}"

        print(message)

        message = encryption(message)

        print(f"{message}")

        client\_socket.sendall(message.encode())

        # Espera la respuesta del servidor

        response = client\_socket.recv(1024)

        response = encryption(response.decode())

        print(response)

    finally:

        # Cierra la conexion

        client\_socket.close()

        return response

def encryption(data):

    key = 'puropinchechensomanalv'

    encrypted = ""

    for i in range(len(data)):

        # Aplica el XOR entre el carácter del mensaje y el carácter de la clave

        # Utiliza la función chr() para convertir el resultado de nuevo en un carácter

        letter = ord(data[i]) ^ ord(key[i % len(key)])

        if data[i] == key[i % len(key)]:

            letter = ord(data[i])

        if letter == 0:

            letter = ord(key[i % len(key)])

        encrypted += (chr(letter))

    return encrypted

#------------------------------------------------------------------------------------------------------------------------------------------

def validate(data):

    # Verifica si los campos estan vacios

    if not data.get():

        messagebox.showwarning("Error", "¨Please, fill all the fields")

        return

    else:

        return data.get()

def create\_users\_window(btn\_function):

    # Crea la nueva ventana

    users\_window = tk.Tk()

    users\_window.geometry("300x150")

    users\_window.title(btn\_function)

    # Crea los elementos de la interfaz de usuario

    label\_username = tk.Label(users\_window, text="User:")

    entry\_username = tk.Entry(users\_window)

    label\_password = tk.Label(users\_window, text="Password:")

    entry\_password = tk.Entry(users\_window, show="\*")

    # Envía a la funcion "send\_to\_server" los datos despues de verificar que los campos no estén vacíos como un parámetro, así como el tipo de acción que realizará

    if btn\_function == "Sign Up":

        button\_action = tk.Button(users\_window, text=btn\_function, command = lambda: [send\_to\_server([validate(entry\_username), validate(entry\_password)], btn\_function)])

    elif btn\_function == "Login":

        button\_action = tk.Button(users\_window, text=btn\_function, command = lambda: [create\_users\_options\_window(validate(entry\_username),

                                                                                                                send\_to\_server([validate(entry\_username),

                                                                                                                                validate(entry\_password)],

                                                                                                                                btn\_function), users\_window),

                                                                                                                                users\_window.destroy()])

    button\_main = tk.Button(users\_window, text="Back", command = lambda: [create\_main\_window(), users\_window.destroy()])

    # Ubica los elementos en la ventana

    label\_username.pack()

    entry\_username.pack()

    label\_password.pack()

    entry\_password.pack()

    button\_action.pack()

    button\_main.pack()

def create\_main\_window():

    main\_window = tk.Tk()

    main\_window.geometry("300x150")

    main\_window.title("Main")

    label\_welcome = tk.Label(main\_window, text="Welcome!")

    button\_login = tk.Button(main\_window, text="Login", command = lambda: [create\_users\_window("Login"), main\_window.destroy()])

    button\_signup = tk.Button(main\_window, text="Sign Up", command = lambda: [create\_users\_window("Sign Up"), main\_window.destroy()])

    label\_welcome.pack()

    button\_login.pack()

    button\_signup.pack()

    main\_window.mainloop()

def create\_users\_options\_window(user, auth, users\_window):

    user\_options\_window = tk.Tk()

    user\_options\_window.geometry("300x150")

    user\_options\_window.title("User Options")

    if user is not False and auth != 'failure':

        label\_greetings = tk.Label(user\_options\_window, text=(f"Greetings, {user}! Please select an option"))

        button\_join = tk.Button(user\_options\_window, text="Join an existing group", command = lambda: [create\_groups\_window("Join Group", user), user\_options\_window.destroy()])

        button\_create = tk.Button(user\_options\_window, text="Create a new group", command = lambda: [create\_groups\_window("Create Group", user), user\_options\_window.destroy()])

        button\_main = tk.Button(user\_options\_window, text="Return to main screen", command = lambda: [user\_options\_window.destroy(), create\_main\_window()])

        label\_greetings.pack()

        button\_join.pack()

        button\_create.pack()

        button\_main.pack()

    else:

        messagebox.showwarning("Error", "Wrong username or password")

        users\_window.destroy()

        user\_options\_window.destroy()

        create\_main\_window()

def create\_groups\_window(btn\_function, user):

    # Crea la nueva ventana

    groups\_window = tk.Tk()

    groups\_window.geometry("300x150")

    groups\_window.title(btn\_function)

    # Crea los elementos de la interfaz de usuario

    label\_groupname = tk.Label(groups\_window, text="Group Name:")

    entry\_groupname = tk.Entry(groups\_window)

    # Envía a la funcion "send\_to\_server" los datos despues de verificar que los campos no estén vacíos como un parámetro, así como el tipo de acción que realizará

    button\_action = tk.Button(groups\_window, text=btn\_function, command = lambda: send\_to\_server([user, validate(entry\_groupname)], btn\_function))

    button\_main = tk.Button(groups\_window, text="Back", command = lambda: [create\_users\_options\_window(user), groups\_window.destroy()])

    # Ubica los elementos en la ventana

    label\_groupname.pack()

    entry\_groupname.pack()

    button\_action.pack()

    button\_main.pack()

create\_main\_window()

# Ejecuta el bucle principal de la ventana

**SERVER:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <signal.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

// Max length for use in strings

#define MAX\_LENGTH 1024

// Max number of users

#define MAX\_USERS 30

// Port for communications

#define PORT 5000

// Code for successful operations

#define SUCCESS 1

// Code for failed operations

#define FAIL 0

/\*

 \* User init format

 \* user name:password:operation

 \* Operation: 1 for login, 2 for signup

 \* Example: admin:root:1

 \* Logins user admin with password root

 \*/

/\*

 \* User key file, stores all users and passwords

 \*/

FILE \*user\_file;

// Key used for XOR encryption

char \*KEY = "puropinchechensomanalv";

/\*

 \* Struct for use with users

 \* Stores the user info along with currently connected socket

 \*/

struct user

{

    int socket; // Socket to which the user is connected

    char \*name; // Username

    char \*pass; // Password

};

/\*

 \* Closes the user key file when ctr+c is pressed whiloe program is running

 \* param sig: Signal, in this case ctrl+c

 \* param \*fp: User key file

 \*/

void sigint\_handler(int sig)

{

    if (user\_file != NULL)

        fclose(user\_file);

    exit(0);

}

/\*

 \* Encrypts and deencrypts a text using a specific key

 \* param data: String to modify

 \* param key: Key used as cipher

 \* return: Modified text

 \*/

char \*XORCipher(char \*data, char \*key)

{

    int len = strlen(data);

    int key\_len = strlen(key);

    char \*output = (char \*)malloc(sizeof(char) \* len + 1);

    for (int i = 0; i < len; ++i)

    {

        int ch = data[i] ^ key[i % key\_len];

        if (data[i] == key[i % key\_len])

        {

            ch = (int)key[i % key\_len];

        }

        if (ch != 0)

        {

            output[i] = ch;

        }

        else

        {

            printf("Null terminator encountered\n");

            output[i] = key[i % key\_len];

        }

    }

    output[len] = '\0'; // Add null terminator at the end

    return output;

}

/\*

 \* Adds a user to the user key file

 \* param user: User name string

 \* param key: Password string

 \* param file: User key file

 \* return: 1 for successful append, 0 for error

 \*/

int add\_user(char \*user, char \*key, FILE \*file)

{

    char buffer[MAX\_LENGTH];

    char userkey[MAX\_LENGTH];

    sprintf(userkey, "%s:%s\n", user, key);

    rewind(file);

    // Change this to only look for user

    while (fgets(buffer, MAX\_LENGTH, file))

        if (strcmp(buffer, userkey) == 0)

            return FAIL;

    sprintf(userkey, "\n%s:%s", user, key);

    fputs(userkey, file);

    return SUCCESS;

}

/\*

 \* Validates user and password in user key file

 \* param user: User name string

 \* param key: Password string

 \* param file: User key file

 \* return: 1 for successful validation, 0 for error

 \*/

int check\_user(char \*user, char \*key, FILE \*file)

{

    char buffer[MAX\_LENGTH];

    char userkey[MAX\_LENGTH];

    char userkey2[MAX\_LENGTH];

    sprintf(userkey, "%s:%s\n", user, key);

    rewind(file);

    // Add validation for error in password

    while (fgets(buffer, MAX\_LENGTH, file))

        if (strcmp(buffer, userkey) == 0){

            printf("Found user\n");

            return SUCCESS;

        }

    return FAIL;

}

int make\_group(char \*user, char \*group)

{

    FILE \*grp\_file;

    char buffer[MAX\_LENGTH];

    sprintf(buffer, "%s.cnv", group);

    if (access(buffer, F\_OK) == 0)

    {

        return FAIL;

    }

    else

    {

        grp\_file = fopen(buffer, "w");

        fclose(grp\_file);

        sprintf(buffer, "%s.usr", group);

        grp\_file = fopen(buffer, "w");

        fputs(user, grp\_file);

        fclose(grp\_file);

        return SUCCESS;

    }

}

int join\_group(char \*user, char \*group)

{

    char fname[1024];

    sprintf(fname, "%s.usr", group);

    if (access(fname, F\_OK) == 0)

    {

        FILE \*fp = fopen(fname, "a+");

        char usep[1024];

        sprintf(usep, "\n%s", user);

        fputs(usep, fp);

        fclose(fp);

        return SUCCESS;

    }

    else

    {

        return FAIL;

    }

}

/\*

 \* Manages the request of a new user, in that case logs them in or

 \* signs them up, should add them then to the active user list.

 \* param req: A string with user init format

 \* param file: User key file

 \* return: 0 for error, 1 for success

 \*/

int manage\_user\_request(char \*req, FILE \*file)

{

    char parts[3][MAX\_LENGTH];

    char buffer[MAX\_LENGTH];

    char user[MAX\_LENGTH];

    char pass[MAX\_LENGTH];

    int dest = 0;

    int j = 0;

    printf("%s\n", req);

    req = XORCipher(req, KEY);

    printf("%s\n", req);

    // Code for split by ':' character

    for (int i = 0; i < strlen(req); i++)

    {

        if (req[i] == ':')

        {

            strcpy(parts[dest], buffer);

            dest++;

            memset(buffer, 0, sizeof(buffer));

            j = 0;

            continue;

        }

        buffer[j++] = req[i];

        buffer[j] = 0;

    }

    strcpy(parts[dest], buffer);

    strcpy(user, parts[0]);

    strcpy(pass, parts[1]);

    int op = atoi(&parts[2][0]);

    printf("%d\n", op);

    switch (atoi(&parts[2][0]))

    {

    case 1:

        return check\_user(user, pass, file);

        break;

    case 2:

        return add\_user(user, pass, file);

        break;

    case 3:

        return join\_group(user, pass);

        break;

    case 4:

        return make\_group(user, pass);

        break;

    default:

        printf("Code error\n");

        return FAIL;

    }

}

int main(void)

{

    struct user user\_list[MAX\_USERS];

    signal(SIGINT, (void (\*)(int))sigint\_handler);

    int nusers = 0;

    int server\_fd, max\_fds, activity;

    int client\_sockets[MAX\_USERS]; // C initializes the values to 0

    struct sockaddr\_in address;

    int addrlen = sizeof(address);

    fd\_set read\_fds;

    char buffer[MAX\_LENGTH];

    for (int i = 0; i < MAX\_USERS; i++)

    {

        client\_sockets[i] = 0;

    }

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

    {

        perror("Socket creation failed\n");

        exit(EXIT\_FAILURE);

    }

    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("Failure binding socket\n");

        exit(EXIT\_FAILURE);

    }

    if (listen(server\_fd, MAX\_USERS) < 0)

    {

        perror("Failure listening\n");

        exit(EXIT\_FAILURE);

    }

    user\_file = fopen("users.key", "a+");

    FD\_ZERO(&read\_fds);

    FD\_SET(server\_fd, &read\_fds);

    max\_fds = server\_fd;

    while (1)

    {

        if (activity = select(max\_fds + 1, &read\_fds, NULL, NULL, NULL) < 0)

        {

            perror("Select Failure\n");

            continue;

        }

        if (FD\_ISSET(server\_fd, &read\_fds))

        {

            int new\_socket;

            if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address,

                                     (socklen\_t \*)&addrlen)) < 0)

            {

                perror("Error accepting connection\n");

            }

            if (recv(new\_socket, buffer, MAX\_LENGTH, 0) < 0)

            {

                perror("Error reading data\n");

                close(new\_socket);

            }

            if (manage\_user\_request(buffer, user\_file) == SUCCESS)

            {

                send(new\_socket, XORCipher("accept", KEY), strlen("accept"), 0);

                FD\_SET(new\_socket, &read\_fds);

                client\_sockets[nusers++] = new\_socket;

            }

            else

            {

                send(new\_socket, XORCipher("failure", KEY), strlen("failure"), 0);

                close(new\_socket);

            }

        }

        for (int i = 0; i < MAX\_USERS; i++)

        {

            int sd = client\_sockets[i];

            if (FD\_ISSET(sd, &read\_fds))

            {

                // Checks if socket was closed

                if (read(sd, buffer, 1024) == 0)

                {

                    close(sd);

                    client\_sockets[i] = 0;

                }

                else

                {

                    if (manage\_user\_request(buffer, user\_file) == SUCCESS)

                    {

                        send(sd, XORCipher("accept", KEY), strlen("accept"), 0);

                    }

                    else

                    {

                        send(sd, XORCipher("failure", KEY), strlen("failure"), 0);

                    }

                }

            }

        }

    }

    fclose(user\_file);

    return 0;

}