CODE

Client Code:

```
#include <iostream>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>
#include <cstring>
#include <mutex>
#include <signal.h>
#include <thread>
#define MAX_LEN 1024
#define S_string_Size 256
using namespace std;


//RC4 Encryption Algorithm Implementation
void rc4_encrypt(char plaintext[], char ciphertext[], char Key[])
{
   unsigned char S[S_string_Size];
   int i = 0,indx;

   // Initialize initial vector
   unsigned char IV[MAX_LEN];
   int iv_st_indx, iv_end_indx;
   for(int i=0;i<MAX_LEN;i++)
         IV[i] = (i+12)%S_string_Size;
   iv_st_indx = 32;
   iv_end_indx = iv_st_indx+S_string_Size;


   // 1) Key Scheduling Algorithm

   // Initialization of State-Vector S
   // with some values
   for (i=iv_st_indx; i < iv_end_indx; i++)
      S[i-iv_st_indx] = IV[i];

   //In the below code (line 43-47), the values in the S array are swapped with the i and j indexes
produced.
   //At this point, the array S is completely initialized to be used in PRGA as input.
   int j = 0;
   for (i = 0; i < S_string_Size; i++) {
      j = (j + S[i] + Key[i % strlen(Key)]) % S_string_Size;
      swap(S[i], S[j]);
   }

   //2) Pseudo random generation algorithm (Stream Generation):
   //After passing through KSA (above code), its output modified array S acts as the input for PRGA.
   //The below code for PRGA outputs a key based on the state of the array S modified by the above
KSA algorithm.
   j = 0;
   i = 0;
   char temp = 0;
```

```
   for(indx=0;indx<strlen(plaintext);indx++)
   {
      i = (i + 1) % S_string_Size;
      j = (j + S[i]) % S_string_Size;
      swap(S[i], S[j]);

      // 3) Below code generates the byte from S by scrambling entries and XORed with plaintext to
generate ciphertext.
      temp = S[(S[i] + S[j]) % S_string_Size] ^ plaintext[indx];
      ciphertext[indx] = temp;
   }
   ciphertext[indx] = '\0';
   return;
}

//RC4 Decryption Algorithm Implementation
void rc4_decrypt(char ciphertext[], char plaintext[], char Key[])
{
   unsigned char S[S_string_Size];
   int i = 0,indx;

   // Initialize initial vector
   unsigned char IV[MAX_LEN];
   int iv_st_indx, iv_end_indx;
   for(int i=0;i<MAX_LEN;i++)
         IV[i] = (i+12)%S_string_Size;
   iv_st_indx = 32;
   iv_end_indx = iv_st_indx+S_string_Size;

   // Key Scheduling Algorithm
   // Initialization of State-Vector S
   // with some values
   for (i=iv_st_indx; i < iv_end_indx; i++)
      S[i-iv_st_indx] = IV[i];

   //In the below code (line 43-47), the values in the S array are swapped with the i and j indexes
produced.
   //At this point, the array S is completely initialized to be used in PRGA as input.
   int j = 0;
   for (i = 0; i < S_string_Size; i++) {
      j = (j + S[i] + Key[i % strlen(Key)]) % S_string_Size;
      swap(S[i], S[j]);
   }

   //After passing through KSA (above code), the modified output array S acts as the input for PRGA.
   //The below code for PRGA outputs a key based on the state of the array S modified by the above
KSA algorithm.
   j = 0;
   i = 0;
   char temp = 0;
   for(indx=0;indx<strlen(ciphertext);indx++)
   {
      i = (i + 1) % S_string_Size;
      j = (j + S[i]) % S_string_Size;
      swap(S[i], S[j]);
```

```
        // Below code generates the byte from S by scrambling entries and XORed with ciphertext to
generate plaintext.
        temp = S[(S[i] + S[j]) % S_string_Size] ^ ciphertext[indx];
        plaintext[indx] = temp;
    }
    plaintext[indx] = '\0';
    return;
}

int eraseText(int);
bool exit_flag = false;
//Threads to send and receive messages from same from process/client.
thread t_send, t_recv;
int client_socket;

// Secret Key
char Key[] = "4569cc7cdeac82874abccb553abde234bdffa349aaa9be234ccdcbab4bad";

// Send message to receiver
void send_message(int client_socket)
{
    while (1)
    {
        char str[MAX_LEN] = "", name[MAX_LEN] = "", ciphertext[MAX_LEN] = "";

        //Taking Receiver's name as input.
        while (strlen(name) == 0)
        {
            cout << "Receiver: ";
            cin.getline(name, MAX_LEN);
        }

            // Taking input message.
        cout << "Msg: ";
        cin.getline(str, MAX_LEN);

        //sending name of the receiver to server
        send(client_socket, name, sizeof(name), 0);

            //Applying RC4 Encryption on message taken as input in variable str and storing the
            //resultant ciphertext in the ciphertext variable
            //The function takes input plaintext and char array to store ciphertext and secret key.
            rc4_encrypt(str,ciphertext,Key);

            //Acknowledging receiver name on the window of sender with the sent ciphertext.
            cout<<"\nKey : "<<Key;
        cout << "\nAck from Server: Cipher text to "<<name<<": ";
        for (int i = 0; i < strlen(ciphertext); i++)
        {
            std::cout << std::hex << (int)ciphertext[i];
        }
            cout<<endl<<endl;
        send(client_socket, ciphertext, sizeof(str), 0);

        //Detaching client and closing its client socket with '#exit' input.
        if (strcmp(str, "#exit") == 0)
        {
```

```
            exit_flag = true;
            t_recv.detach();
            close(client_socket);
            return;
        }
    }
}

// Receive message
void recv_message(int client_socket)
{
    while (1)
    {
        if (exit_flag)
            return;
        char name[MAX_LEN] = "", str[MAX_LEN] = "", plaintext[MAX_LEN] = "";

        //Receiving name of the sender from the server.
        int name_bytes = recv(client_socket, name, sizeof(name), 0);
        if (name_bytes <= 0)
            return;
        eraseText(30);
        cout << endl;

        //Receiving message of the sender from the server.
        int message_bytes = recv(client_socket, (unsigned char *)str, sizeof(str), 0);

        // if name of the sender is present.
        cout<<"\nKey : "<<Key;
        if (strcmp(name, "#NULL") != 0)
        {
                //Printing the name of the sender.
            cout << "\nCipher text from " <<name<<": ";
            for (int i = 0; i < strlen(str); i++)
            {
                 printf("%02x", str[i]);
            }
            cout<<endl;
            cout << endl<<endl;

            //Applying RC4 Decryption on ciphertext taken as input in variable str and storing the
                //resultant decrypted message in the plaintext variable.
                //The function takes input ciphertext sent from server in str, decrypted message in the
variable plaintext
                //and secret Key.
            rc4_decrypt(str,plaintext,Key);

            //Printing sender's name with his/her decrypted message.
            cout<<name<<": ";
                cout<<plaintext;
            cout <<endl<<endl;
        }
        else
            cout << str << endl;
        fflush(stdout);
    }
}
```

```
int main()
{
    //Creates socket and connects to the server.
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_addr.sin_port = htons(50201);

    if ((connect(client_socket, (struct sockaddr *)&server_addr, sizeof(struct sockaddr_in))) == -1)
    {
        perror("connection error.");
        exit(-1);
    }
    char name[MAX_LEN];

    //Taking the name of the new user/client.
    cout << "User-Name: ";
    cin.getline(name, MAX_LEN);
    send(client_socket, name, sizeof(name), 0);
    cout<<"\n********** "<<name<<" Chat-box ************";
    cout << endl;

    //Using threads to send and receive messages for the same process or client.
    thread t1(send_message, client_socket);
    thread t2(recv_message, client_socket);

    t_send = move(t1);
    t_recv = move(t2);

    if (t_send.joinable())
        t_send.join();
    if (t_recv.joinable())
        t_recv.join();
    return 0;
}

//To take care of the spaces broadcasted by server.
int eraseText(int cnt)
{
    char back_space = 8;
    for (int i = 0; i < cnt; i++)
    {
        cout << back_space;
    }
    return 0;
```

Server Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <thread>
#include <mutex>
#define MAX_LEN 1024

using namespace std;

const int MAX_CLIENTS = 5;
//Structure to define a client.
struct terminal
{
	int id;
	string name;
	int socket;
	thread th;
};

//Creating vector of clients.
vector<terminal> clients;

int seed = 0;
mutex cout_mtx, clients_mtx;

void set_name(int id, char name[]);
void broadcast_message(string message, string clientMessage, int sender_id);
void end_connection(int id);
void handle_client(int client_socket, int id);

int main()
{
	int server_socket;
	//Creating socket from server.
	if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == -1)
	{
		perror("socket: ");
		exit(-1);
	}

	struct sockaddr_in server;
	server.sin_family = AF_INET;
	server.sin_port = htons(50201);
	server.sin_addr.s_addr = INADDR_ANY;

	//Binding server socket.
```

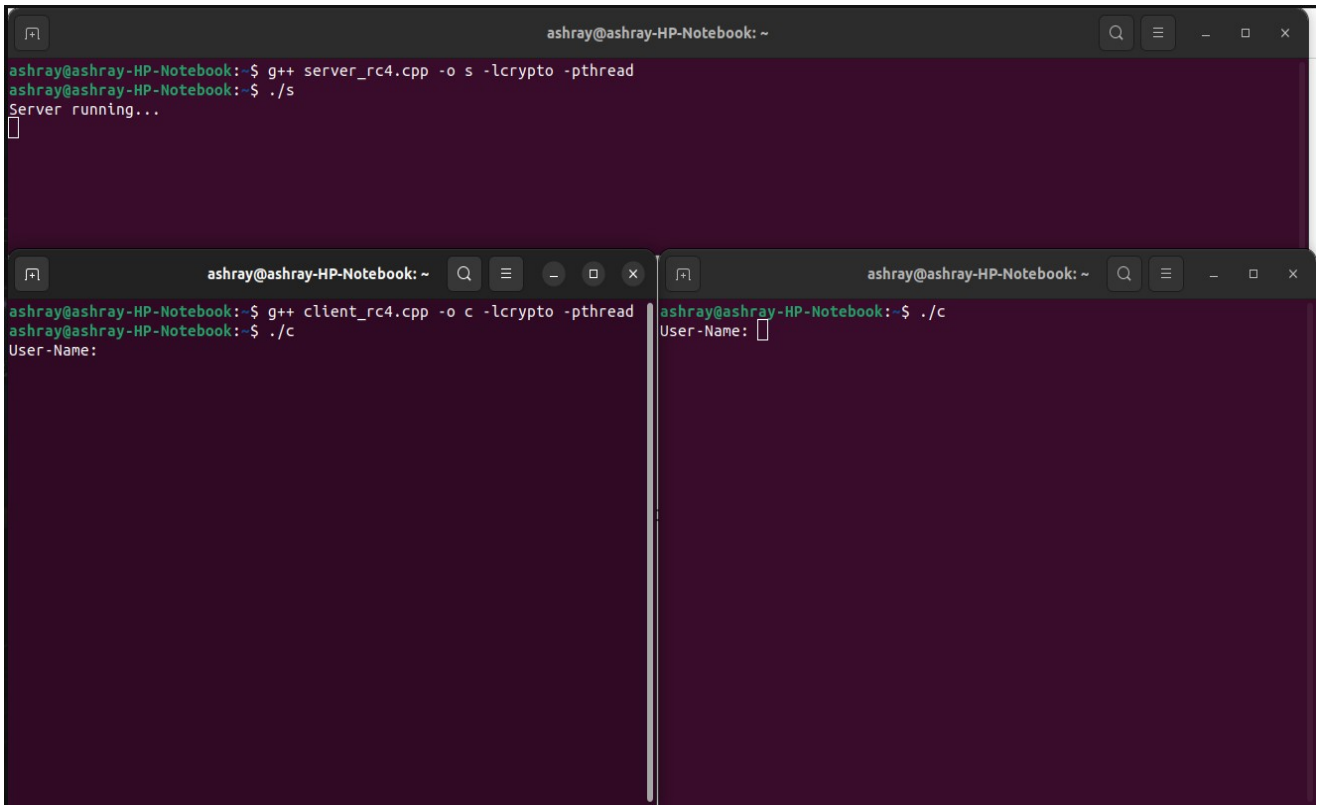
```
	if ((bind(server_socket, (struct sockaddr *)&server, sizeof(struct sockaddr_in))) == -1)
	{
		perror("bind error: ");
		exit(-1);
	}

	//Listening on the assigned port.
	if ((listen(server_socket, MAX_CLIENTS)) == -1)
	{
		perror("listen error: ");
		exit(-1);
	}
	cout << "Server running..." << endl;

	struct sockaddr_in client;
	int client_socket;
	unsigned int len = sizeof(sockaddr_in);

	//Running and infinite loop to take connections from different clients.
	while (1)
	{
		if ((client_socket = accept(server_socket, NULL, NULL)) < 0)
		{
			perror("accept error: ");
			exit(-1);
		}
		seed++;
		thread t(handle_client, client_socket, seed);
		lock_guard<mutex> guard(clients_mtx);
		clients.push_back({seed, string("newUser"), client_socket, (move(t))});
	}

	// Joining different client's threads.
	for (int i = 0; i < clients.size(); i++)
	{
		if (clients[i].th.joinable())
			clients[i].th.join();
	}

	close(server_socket);
	return 0;
}

// Helping function to set the name of client by first finding if the
// the current client structure by id and assigning its name attribute with the provided name.
void set_name(int id, char name[])
{
	for (int i = 0; i < clients.size(); i++)
	{
		if (clients[i].id == id)
		{
			clients[i].name = string(name);
		}
	}
}

// Function to send message to the specific client with the name provided as the receiver.
```

```
void broadcast_message(string message, string clientName, int sender_id)
{
	char temp[MAX_LEN];
	strcpy(temp, message.c_str());
	if (clientName.length() > 0)
	{
		// Finding the name of the client same as that provided as receiver and sending the
input message.
		for (auto &i : clients)
		{
			if (i.name == clientName)
			{
				send(i.socket, temp, sizeof(temp), 0);
				return;
			}
		}

	}
	// Else broadcasting to all the clients in the system.
	else
	{

		for (int i = 0; i < clients.size(); i++)
		{

			if (clients[i].id != sender_id)
			{
				send(clients[i].socket, temp, sizeof(temp), 0);
			}
		}
	}
}


void handle_client(int client_socket, int id)
{
	char name[MAX_LEN], str[MAX_LEN], clientName[MAX_LEN];

	//Receiving sender's name in the variabe 'name'.
	recv(client_socket, name, sizeof(name), 0);
	set_name(id, name);


	string welcome_message = string(name) + string(" has joined");

	while (1)
	{
		// Receiving receiver's name in the variable 'clientName' and ciphertext from sender
		// in the variable str.
		int name_received = recv(client_socket, clientName, sizeof(clientName), 0);
		int bytes_received = recv(client_socket, str, sizeof(str), 0);
		if (bytes_received <= 0)
			return;

		// Printing sender's name, receiver's name and the ciphertext sent by sender to receiver.
		cout<<"From "<<name<<" to "<<clientName<<" Cipher-text: ";
		for (int i = 0; i < strlen(str); i++)
```

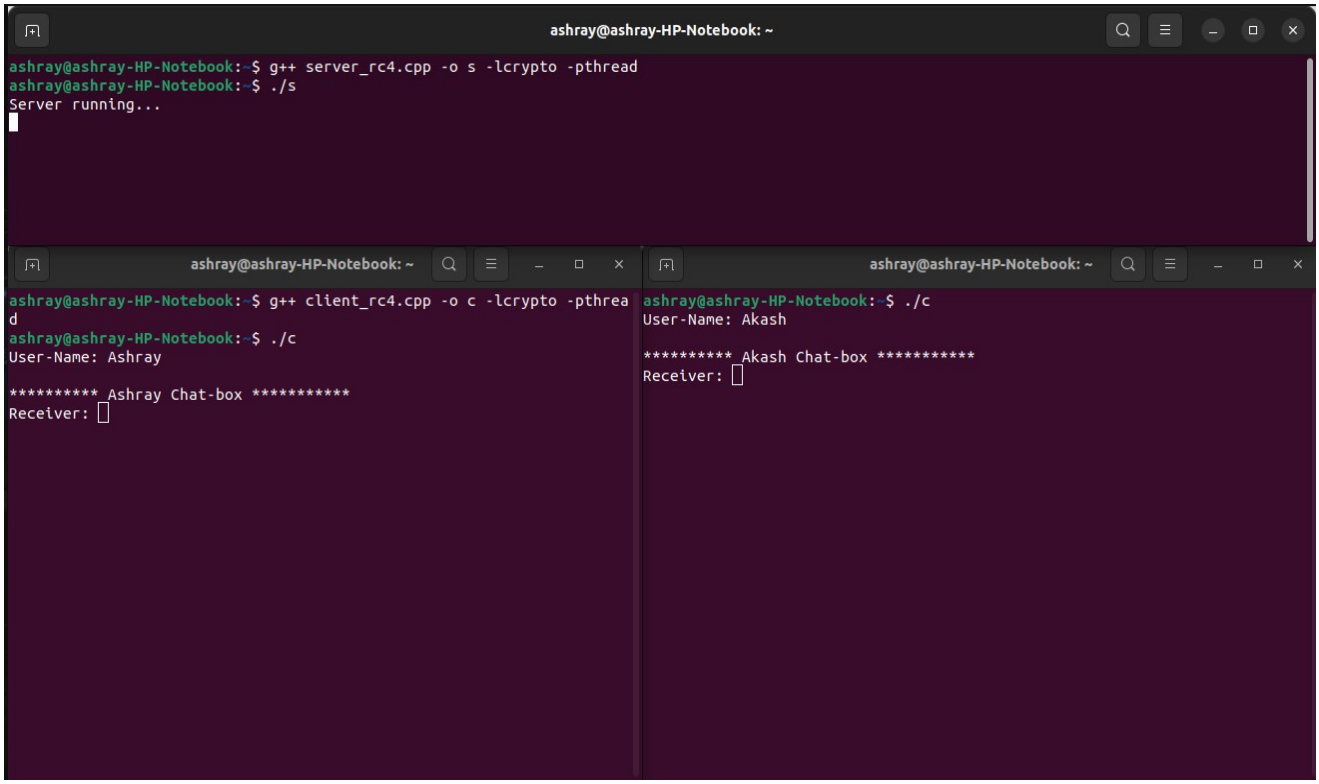
```
                {
            std::cout << std::hex << (int)str[i];
                }
                cout<<endl;
                if (strcmp(str, "#exit") == 0)
                {
                        string message = string(name) + string(" has left");
                        end_connection(id);
                        return;
                }
                // Sending sender's name to the receiver.
                broadcast_message(string(name), string(clientName), id);

                // Sending ciphertext received from sender to the receiver.
                broadcast_message(string(str), string(clientName), id);
        }
}

// Ending connection and detaching the threads of the client.
void end_connection(int id)
{
        for (int i = 0; i < clients.size(); i++)
        {
                if (clients[i].id == id)
                {
                        lock_guard<mutex> guard(clients_mtx);
                        clients[i].th.detach();
                        clients.erase(clients.begin() + i);
                        close(clients[i].socket);
                        break;
                }
        }
}
```

OUTPUT

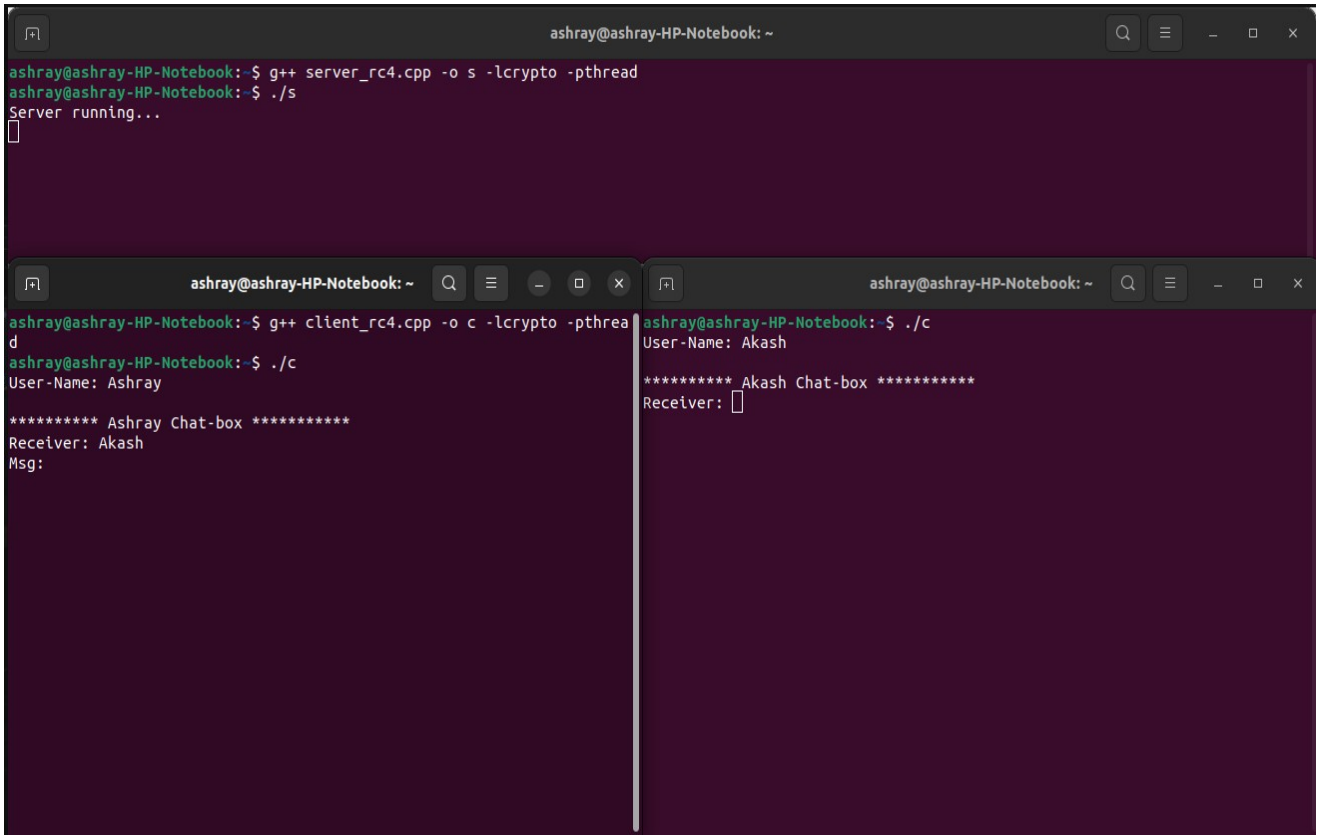
1. Running Server on one terminal and 2 Clients on two different terminals.

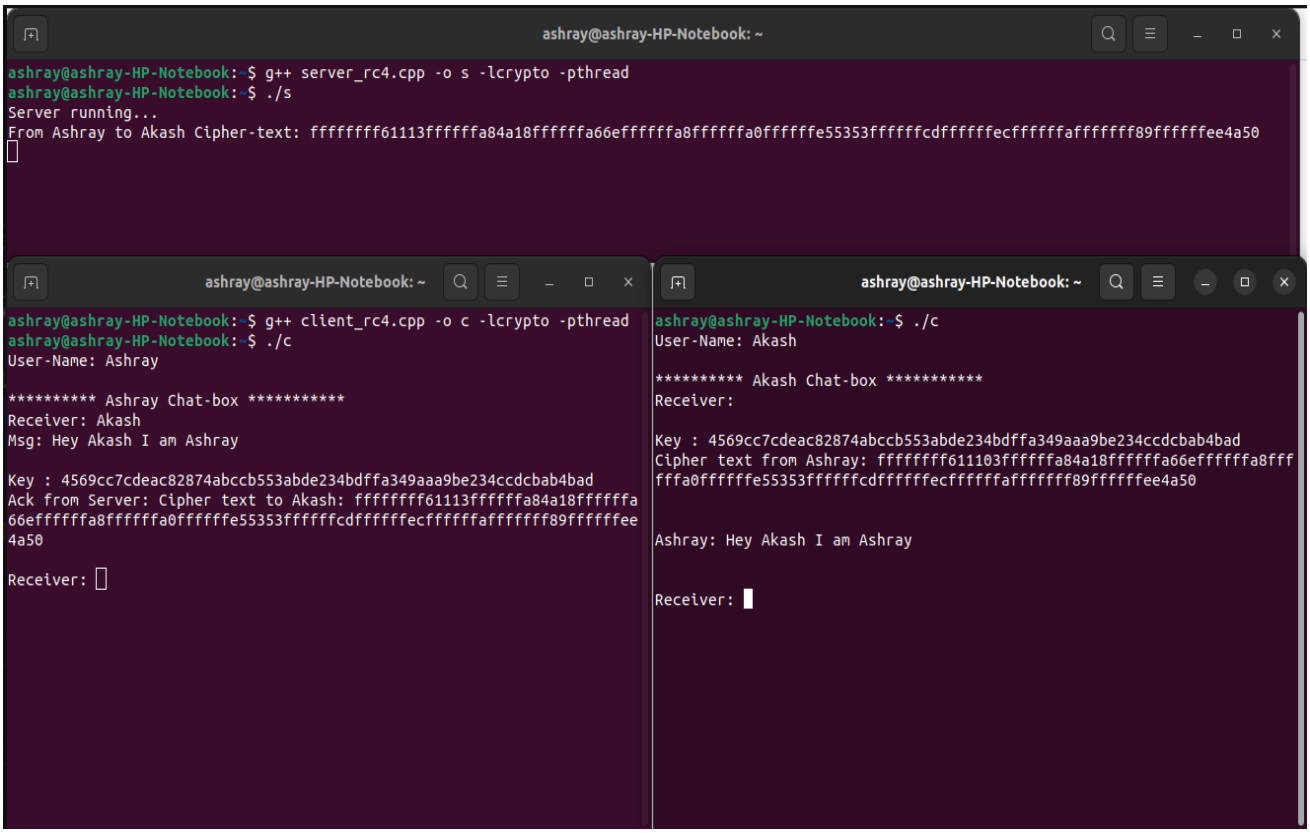


2. Adding User's chat boxes by providing names to each client.



3. Sending message to user ‘Akash’ by user ‘Ashray’. Prints Key used and acknowledgement from server.





4. Sending message from user ‘Akash’ to user ‘Ashray’ and printing Key and ciphertext.