COE817

ANONYMOUS MESSAGE BOARD:
DESIGN & IMPLEMENTATION

April 19, 2017

Group Members: Rachel Lan Chung Yang, Nicholas Giamblanco, Sean True, Ryan Murari
Student #: 500452073, 500551269, 500579636, 500594865
Ryerson University
Department of Electrical & Computer Engineering

Contents

1.0 Abstract

Anonymous message broadcast is a scheme based on the Dining Cryptographers Problem, enabling its users to communicate to one another without revealing their identity. More specifically, the participants can be aware of who takes part in the conversation but will not be able to know from who the messages originate from. Socially, such a system can allow for care free expression among the participants, for example, in the case of meeting between a manager and its employees where they do not have to fear unfair consequences if they express an opposed opinion to their manager. The manager may be able to take a guess but will be unable to demonstrate the exact origin of the message. Another application of such a system can be seen in online free-to-join chatrooms, allowing strangers to communicate anonymously without revealing their personal information's.

2.0 Introduction

This project implements a system of anonymous message broadcast. The project is based on socket programming and the newly developed stream cipher G-Cipher is split in two components: the Server software and the Client software. The G-Cipher is a stream cipher developed by Nicholas Vincent Giamblanco in 2017 which will be described in details later in this report.

The goal is to create a chat room to which participants can connect, check users that are currently active and exchange transmissions with no possibility of knowing the specific sender of the messages. The base scenario assumed in this project will be the case of the manager having meeting with his employees.

In the context of this application the clients are any users that wishes to enter the chatroom and discuss anonymously (i.e. manager and employees). All clients are required to have knowledge of their login information as in, their username and the group password. Once they enter the room, the user should then be able to enter messages that will be sent to all other active users and read incoming messages. The client software is also equipped with the tools to make sure his communications cannot be intercepted. Making use of the latest version of G-Cipher to encrypt the communications with the server, users are protected from eavesdropping as well as man-in-the-middle attacks as a nonce is also encrypted along side the data. His available commands are displayed during the initial login or with the

-help command, permitting them to display the currently connected users using -ls, or logout using -quit.

To enable the chat room to be join-able by the clients, a server software must be running on another machine. The server's role is simply to forward the packets to be exchanged during the discussion. The server is notified upon the joining of a user and can as well keep track of the current database of users and currently active ones.

This message broadcast service which allows anonymous discussions is based on the Dining Cryptographer Problem which discusses how to perform secure multi-party computation of the Boolean-OR function. Proposed in the early 80s, the problem presents three cryptographers gathered around a dinner table. The waiter informs them that dinner has already been paid by someone, who could be one of the cryptographers or National Security Agency. The cryptographers respect each other's right to make an anonymous payment, but want to find out whether the NSA paid. So, they decide to execute a two-stage protocol.

In the first stage, every two cryptographers establish a shared one-bit secret, say by tossing a coin behind a menu so that only two cryptographers see the outcome in turn for each two cryptographers. Suppose, for example, that after the coin tossing, cryptographer A and B share a secret bit 1, A and C share 0, and B and C share 1.

In the second stage, each cryptographer publicly announces a bit, which is:

- if they didn't pay for the meal, the exclusive OR (XOR) of the two shared bits they hold with their two neighbours
- if they did pay for the meal, the opposite of that XOR.

Supposing none of the cryptographers paid, then A announces 1XOR0 = 1, B announces 1XOR1 = 0, and C announces 0XOR1 = 1. On the other hand, if A paid, he announces NOT (1 XOR 1) = 0.

The three public announcements combined reveal the answer to their question. One simply computes the XOR of the three bits announced. If the result is 0, it implies that none of the cryptographers paid (so the NSA must have paid the bill). Otherwise, one of the cryptographers paid, but their identity remains unknown to the other cryptographers. Cryptographers respect each other's right to make an anonymous payment, but want to find out whether the NSA paid. So, they decide to execute a two-stage protocol.

3.0 Project Specifications

The project consist of creating an anonymous message board. In this application, the server has been pre-configured with a set of users and a common group password. The users are assumed to know their usernames and the group password prior to starting the application. They must input these pieces of information to access the message board. Messages cannot be sent until every member have logged into the message board. Once all the users are present, the discussion can commence, but the message relays must remain anonymous; it should not be possible to identify the author of any of the messages. Users, however, should be able to see who is present in the message board.

4.0 Project Design

Through the use of fundamental concepts, such as socket programming, stream ciphers, and algorithmic logic, the anonymous message board was created. A detailed explanation of these concepts and the design process will be explained throughout this section of the report.

4.1 Client Server Model

The client server model consist of a relationship between one or more client programs and a server program. This relationship can occur due to socket programming. Bound to a port number, sockets are the end points of a two way communication between programs. When port numbers are used in combination with IP addresses, this allows the communication to occur across networks.

In this project, the client program merely displays a user interface. There, the user can input information which will be relayed to the server program. The server then performs the request and relays a response to the client. For example, during the login process, the user will input their username and password in the client application. This information will be validated by the server, and the server will respond with either an error message, if the information is invalid, or by allowing the user to access the message board, if valid. The client program provides a transparent user experience, the encryption of every message, prior to sending it to the server program is instantaneous, a seamless user experience.

The server program is designed to monitor the authenticity of the users by matching the inputted usernames and passwords to the ones on the system. Once all the users have entered the

chatroom, it also broadcasts any incoming message to each participant and monitors the users' presence.

4.2 Network Security

Key elements of a secure network defined by a system's ability to provide authenticity, confidentiality, and integrity. Authenticity is defined by the ability of determining and confirming the identity of an individual on the system. Confidentiality is a mean of protecting information from any unauthorized party and protecting the privacy of its users. Lastly integrity is defined by the system's ability to provide data which is neither altered, nor destroyed by any unauthorized means. All three of these components have been integrated into the chatroom application.

Authenticity has been created through the user's login process. Users must enter their username and the group password in order to enter the chat. If these pieces of information do not match with the information stored on the server, the user will not gain access to the chatroom. In addition, the same user cannot be logged into the chatroom more than once. This prevents any unauthorized party to try to replay information sent to the server to gain access to the message board.

Confidentiality and integrity were created through data encryption. Encryption is a way of masking messages such that only authorized parties can understand the present information. For this application, the G-cipher, a stream cipher was to encrypt the messages passed between the client and server. The former consist of performing a set of pseudo-random operations upon every digit of a data stream. A detailed explanation of the G-cipher has been outlined in the implementation section of this report.

The message board also utilizes nonces in order to increase the security of the application. Nonces are pseudo-random values which can only be used once. This allows the server application to know whether an attacker has tried to intercept and re-transmit a message. If the server receives messages with the same nonce, the messages can therefore be ignored and will free up some of its resources.

However there are some security features that were not included as they were extensive for the scope of this application; session keys, for example, were never regenerated. In theory, session keys should be periodically regenerated in order to ensure their confidentiality. Since this application was

not designed for extensive use, it was deemed unnecessary for the scope of this project. It was also determined that public key cryptography was not required. Public keys are displayed to the public and are often pared with a private key in order to validate the user's identity. Yet the project description clearly states that the server should be pre-configured with the group password and usernames. It could thus be assumed that the users must be aware of this information and with those tidbit of information, it should be sufficient to validate their identity. Similarly, IP headers were not encrypted and the Internet Protocol Security (IPSec) were not used. It is important to note that when more security features that are introduced to an application, the system's performance will start y to decrease. The implemented security features were though to be sufficient for the scope of this project.

4.3 Anonymity

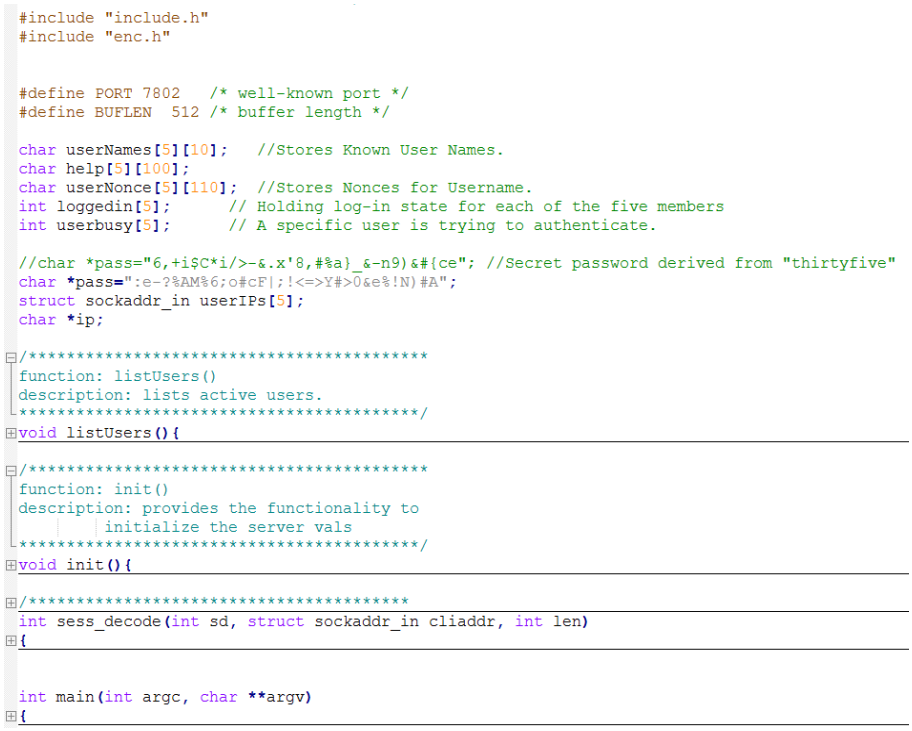
One of the key features of the message board is that every message being sent remains anonymous. Thus, when a user inputs a message through the client program, the same message is sent to the server. The server does not store the message, but simply changes the user's name to an anonymous tag, An0n, and broadcasts the message to the other participants of the group chat. The user, however is aware of the messages that it has sent which are identified on their user interface. It is, therefore, impossible to distinguish the author of a message unless a third party gains access to the author's user interface. However, due to the encryption method described above, this becomes highly unlikely.

5.0 Implementation

In order to implement the intended design, three C program files were created. These files are client.c, which describes the client-side software, server.c, which describes the server-side software, and enc.c, which handles encryption, decryption and authentication. Additionally, the include.h file is used to define various parameters and constants, as well as the PDU structure. In order to use this application, the server.c file must first be run on a server terminal. This file opens a port and begins to accept connections. Then, client terminals may begin to connect to the server via the client.c file. Should the clients submit correct information to the server, they can be authenticated by the server and

participate in the chat. The chat procedures begin when all users are connected to the server; any messages sent prior to all users being present are ignored.

Communication between the server and clients is done primarily through use of PDU structures. These structures are defined as “pdu_t” structures in the file include.h. These structures contain three fields: type, sname, and data. The type field specifies the purpose and contents of the PDU. There are 11 types of PDU: MSG, which specifies a chat message; BCAST, which specifies that the PDU is being broadcast to all clients; LST, which requests or specifies a list of all currently active users; E, which specifies an error message; QUIT, which requests a disconnection from the receiver; INFO, which sends server information to users; HELP, which requests and sends help information; AUTH, which indicates a successful authentication; BAD, which signifies invalid authentications; CHAL, which denotes a challenge packet; and RESP, which indicates a challenge response packet. The next field in the PDU structure is sname, which is the sender name. Should a client send a packet, it is the client’s username. If the server sent the packet, it is the server’s name. However, all messages relayed by the server have the name “An0n” in their sname field. This is to preserve the initial sender’s anonymity. Finally, the data field contains the payload of the PDU. This is used to carry information between the clients and server.



```
#include "include.h"
#include "enc.h"


#define PORT 7802   /* well-known port */
#define BUFLEN  512 /* buffer length */

char userNames[5][10];   //Stores Known User Names.
char help[5][100];
char userNonce[5][110];  //Stores Nonces for Username.
int loggedin[5];      // Holding log-in state for each of the five members
int userbusy[5];      // A specific user is trying to authenticate.

//char *pass="6,+i$C*i/>-&.x'8,#%a}_&-n9)&#{ce"; //Secret password derived from "thirtyfive"
char *pass=":e-?%AM%6;o#cF|;!<=>Y#>0&e%!N)#A";
struct sockaddr_in userIPs[5];
char *ip;

/*****************************************
function: listUsers()
description: lists active users.
*****************************************/
void listUsers(){

/*****************************************
function: init()
description: provides the functionality to
        initialize the server vals
*****************************************/
void init(){

/*****************************************
int sess_decode(int sd, struct sockaddr_in cliaddr, int len)
{


int main(int argc, char **argv)
{
```

Figure 1: server.c functions and variables

5.1 server.c

Figure 1 shows the functions and variables in the server.c file. The server.c file contains 4 functions: main(int argc, char**argv), listUsers(), init(), and sess_decode(int sd, struct sockaddr_in cliaddr, int len). When the application is first run, the init() function is called within the main function. This function initializes all array values in the file. This includes initialization of the username values and the help menu strings, which store the list of valid usernames and the help information that the users may request, respectively. It also initializes the users' statuses, marking them as being logged off and unoccupied. It then attempts to create and bind a TCP socket. Should it fail, the program is exited. If the socket is successfully bound, the application then listens for 5 connection requests. It then performs sess_decode, which parses received messages from the client. The sess_decode function takes in a PDU from a client and uses the type field to decide the course of action to take. There are 6 kinds of messages to parse, each depending on the PDU type. When the client is successfully authenticated, it sends an authentication message to the server, who then broadcasts the new user to all others. Once the user is authenticated, the server marks their status as being logged in. While in the process of authentication, it changes their state to "busy". These states assist in determining if a challenge request is valid. Challenge messages represent authentication requests, and any valid authentication requests are responded to. This process has the server check the selected user's status; if they are busy or logged in, the challenge is invalid, as the user is already present. Additionally, an unknown user cannot log in, and their challenge request is invalid. If the user is known, and not already present, the challenge is valid. Message type PDUs are chat messages sent by clients; when the server receives a message, it replaces the sender field to preserve anonymity and sends the encrypted message to all other clients. However, it will only respond to messages other than authentication and challenges once all users have logged in. Messages sent when not all users are present are not relayed to other logged-in users. This is because having fewer users in the server may compromise anonymity, as having fewer participants can result in users being able to figure out who is sending messages. A list PDU is a request by a user for a list of all active users, thus the server sends the names of all currently-active users to the original sender in response. Receiving a help message results in the server sending the help messages to the client. Finally, a quit message signals to the server that the client wishes to leave the chat room. It resets the user's status in the server, and notifies the other user that they have left.

```
//Global Variable Definitions

    char name[10]; //; Client User Name
    char key[32]; //; Client/Server Session Key
    int reg=0, done=0;
    int sockfd; //; Socket Descriptor
    struct sockaddr_in servaddr,cliaddr;

//Thread Mutex
    pthread_mutex_t mutexsum = PTHREAD_MUTEX_INITIALIZER;

//NCURSES definitions for windows...
    int parent_x, parent_y, new_x, new_y;
        int in_winsize = 4; //3
    int line=1; // Line position of top
    int input=1; // Line position of top
    WINDOW *msgwin;
    WINDOW *inputwin;


/********************************************************************
function: sendMessage()
description: updates screen and sends message to server Used by a thread.
********************************************************************/
void *sendMessage(){

/************************************************************
function: listener()
description: updates screen and responds to a server update.
************************************************************/
void *listener(){




/*********************************
Main() of client.c
*********************************/

int main()
{
```

Figure 2: client.c functions and variables

5.2 client.c

The client-side software defined in client.c is shown in Figure 2. This file contains three funtions: main(), listener(), and sendMessage(). When beginning, the client attempts to create a socket connection with the server, and initializes all of the variables. It then requests the user's username and password. Using this information, it creates a challenge PDU with a randomly generated nonce, and sends it to the server. It then parses the response. The server may respond by accepting a valid challenge, or by refusing an invalid response. If the server accepts the challenge, the client then constructs an authentication packet, which is sent to the server. If the server also responds with an authentication packet, the client can join the chat room. It then sets up an interface, and then creates

two threads using the listener() and sendMessage() functions. The sendMessage() thread gets user input and constructs an encrypted PDU, and sends it to the server. This PDU can be a HELP, LST, or MSG type. The listener() thread, meanwhile, listens on the opened socket. It gets information sent by the server and displays it on the screen. These two threads provide all necessary chat room functionality on the client terminals.

5.3 enc.c

The final file is the enc.c file. This file contains all functions needed to implement the encryption cipher, the G-cipher. This file generates vectors for use as keys in encryption and decryption from the entered password. Should a password entered by a user be correct, the vector generated will match the one used for encryption and decryption on the server. It does this using the passEnforce function; this function seeds a random number generator using the password values, and uses the numbers generated from it to form the key. By seeding the generator, the numbers that it outputs become predictable. The enc.c file contains functions for encryption and decryption, which are used whenever information needs to be sent over a socket. This encryption uses the generated key to generate pseudo-random numbers in order to obfuscate the inputted text. This can be reversed by performing inverse operations on the obfuscated data in order to find the original contents. These encryption and decryption functions are used in the clients and the server.

6.0 Results

The following consist of a few screen captures of the application. Displaying the error handling procedure of the application as well as the application running successfully.

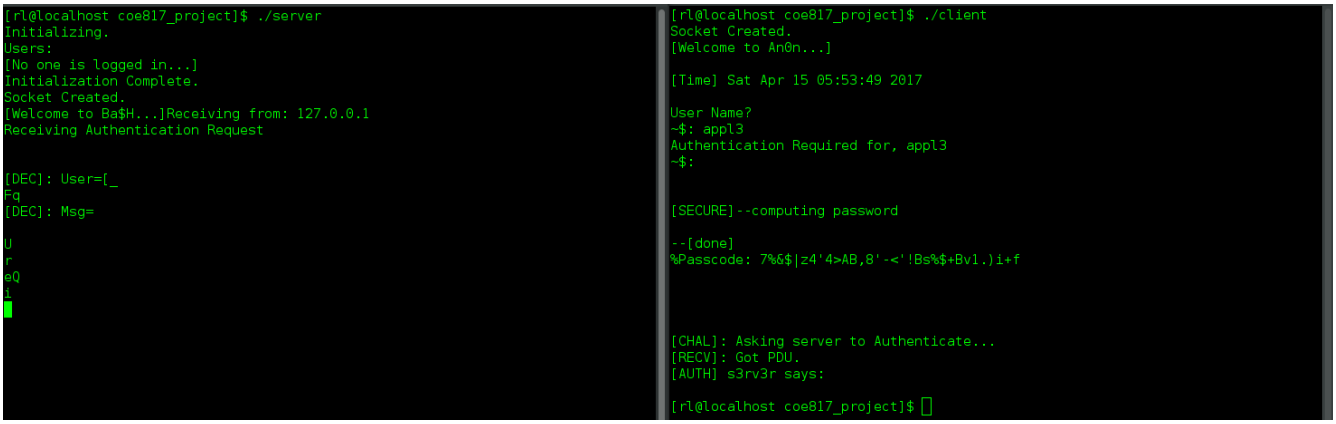


Figure 3: Unsuccessful Login – Wrong password
(Left—Server program, Right—Client program)

Figure 4: Successful Login
(Left—Server program, Right—Client program)

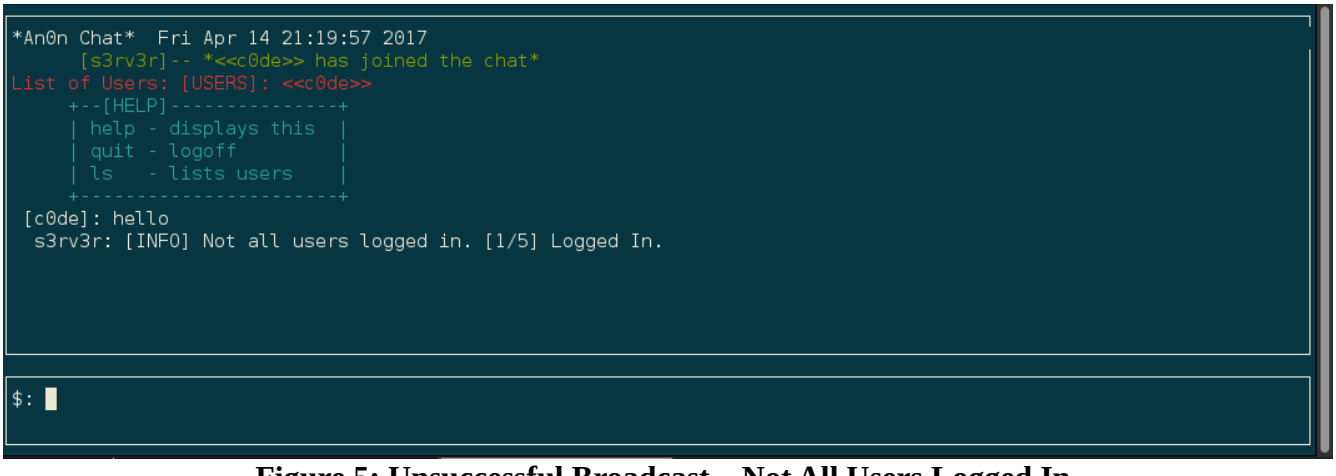


Figure 5: Unsuccessful Broadcast – Not All Users Logged In
(Client Program)

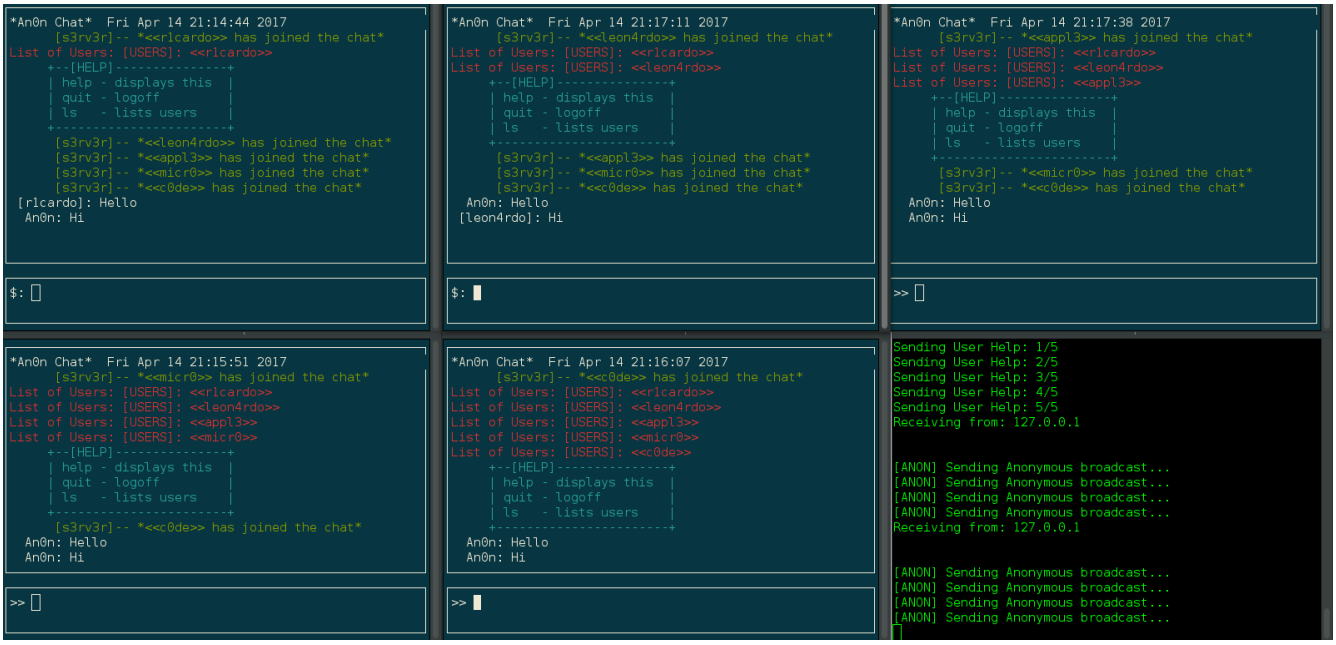


Figure 6: Successful Anonymous Broadcast
(Bottom Right—Server program, All other windows—Different Client Program Instances)

7.0 Conclusion

In conclusion, this application demonstrated that anonymity can be maintained through a server client model. The implementation used Network security theories such as the stream cipher and cryptographic nonces, socket programming, and application logic. The application accurately depicted message board which conserved the identity of its users and was tested various ways to ensure the accuracy and efficiency of the system.