Department of Electrical and Computer Engineering

University of Massachusetts Dartmouth

---ECE549 Network Security---

Team Lead: Cameron Whittle
Members: Peter McGrory, James McCarthy, Devaj Ramsamooj

Table of Contents

1. Abstract

This project examined the difficulty of adding password protection as well as RSA based encryption to messages in Amity Communications, and how that encryption makes the overall system more resilient to attackers. Amity Communications is a group chat messaging system designed to reduce clutter and maintain organization by utilizing three separate chat boxes. Once RSA encryption was added, the server was examined to ensure that the messages were truly being encrypted efficiently. We will analyze and demonstrate how to manage implementing encryption into a TCP based group chat messenger, and how to securely and effectively handle key distribution in said architecture.

2. Introduction

2.1 Purpose

There are 124.5 billion emails and 6 billion texts sent each day in the United States alone. These messages range from work related emails, to personal texts between friends, and much more. What do all these messages have in common? Each message must be sent across unsecure networks in order to reach its destination. This is where attackers and hackers come into the picture. With unprotected data, an attacker can easily get ahold of information that shouldn't be shared. Encryption algorithms are an attempt to protect a person's data from these attacks. By encrypting data, attackers won't be able to read what you are saying, even if they get ahold of certain pieces of the information required. Now a days, basically all data has some form of cyber protection, and for good reason.

Last semester, in the Spring of 2019, our team built a group chat messenger known as Amity Communications. This chat service prioritized organization in often cluttered and busy group chats, featuring 3 separate chat boxes per chat room. Along with this, the system prioritized simplicity, designed for students with busy work and social lives. It was designed on a TCP framework and was built in Windows Forms using the C# language. The only issue was that, given our limited knowledge at the time, it had no implemented security measures. Anyone using an application like Wireshark, could see the plaintext data being sent in the messenger. Our plan, was to upgrade the system, creating Amity Communications v2.0, a more secure, and encryption protected group chat messenger. In doing this, we could study, and learn firsthand the way security can be implemented into a chat messenger, and continue to build off a previous project, creating a safer, stronger, and better system.

2.2 Goals

To achieve the final overall goal, the team divided the tasks into a list of goals that needed to be completed for the system to be successful. The first goal was to analyze the different encryption methods available and choose the one that best suited the project. Below is a basic flow diagram showing how the system should function in the end, with the new encryption added. The actual method used is more complicated than what is shown in this diagram and will be explained more in section 5.2.

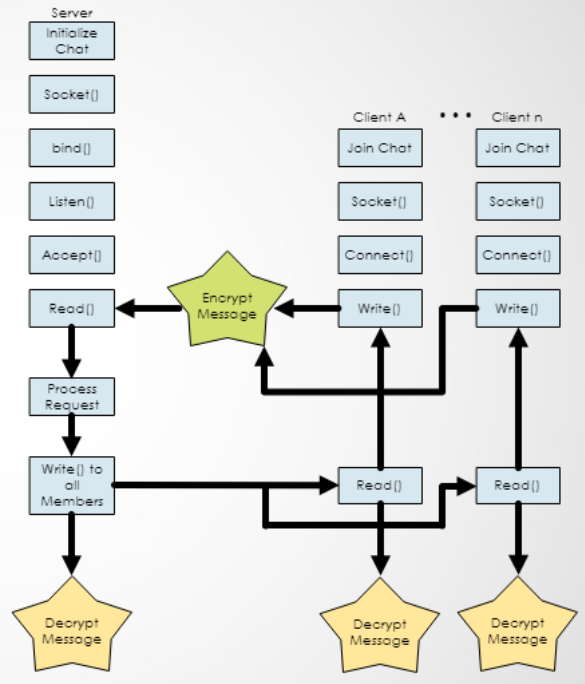

Figure 1: System Flow Diagram

The next step was to design a way to handle encryption keys. The ways that the different members and server handle their keys and how they are used is crucial to how cryptosystems operate. The methods used must be smart enough so that an attacker can't easily intercept or obtain one of these keys used for decryption. Lastly, the team needed to implement these into Amity Communications, creating v2.0 and showcase its overall effectiveness.

3. Problem Definition

3.1 Scope

The scope of this project, from a basic standpoint, was to implement security measures into our older version of Amity Communications. This would be accomplished through an

encryption algorithm, as well as a key distribution method. This application's utilized security information has been covered throughout the semester, giving our team the chance to bring what we have learned in class into the real world through a group messenger program. Also falling into the scope of the project, we wanted to retain the same appeal that came from version 1.0, where the user has a clear view of all incoming messages.

3.2 C.I.A.

Amity Communications now has total Confidentiality over its users by having the user encrypt their own message before sending it to the server, preventing any attackers trying to sniff packets being sent. The messages withhold Integrity when being sent between all active users. This can be shown by viewing the ciphertext on the client side, then comparing with the server outputs of the received message. To help with the Availability for each user, a simple username/password was created for each group member as an added security measure.

3.3 Claim

Amity Communications v2.0 will apply the RSA algorithm to give users some much needed security. The ciphertext from user messages will be accessible through the server output, allowing the owner to obtain a log off all incoming messages. The market value for this software is simply open source.

4. State of the Art & Challenges

4.1 Current State of the Field of Research

Ending last semester, Amity Communications had the ability for multiple users to join a chat server over a TCP connection. While connected each user could communicate with one another, utilizing shared specified chat rooms. Although, in these chatrooms all messages are sent in plaintext, throwing any security out the window. The image below is an example of the server output when receiving a plaintext message.

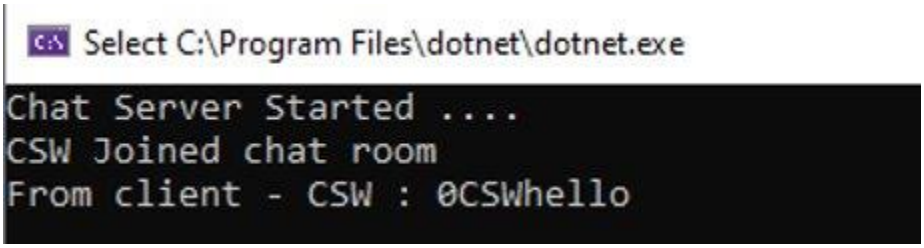

Figure 2: ACv1.0 Server Example

4.1 What Sets This Project Apart

This specific project interested the group through the opportunity to go back and work again on something from the past, and to add brand new features. This project tested our groups capabilities with C#, client and server interactions, TCP connections, group chats, RSA encryption, and key distribution.

4.3 Issues Faced

A big issue that our team had from the start was adding the actual encryption algorithm to our existing program. In ECE 369 our team did not have encryption in mind when first

making this project, and we had left our code without any intentions of working on the same project this semester. This all lead to dedicating a fair amount of time to relearning and rewriting some of the older sections of code. Another issue we kept running into was the interaction of public/private keys between the server and client (this can be viewed in section 5.2). However, these issues were eventually conquered, ads the final deliverable functioned effectively.

5. Work Conducted

5.1 Amity Communications v2.0

The first thing that the team implemented was the user profiles sub system. This served more as a proof of concept, however it was quite functional and adds an extra layer of security onto the system. Each user has his/her initials stored in the system, along with a password of their choice. In order to join the chatroom, the user must login. If the credentials match, then the TCP connection with the server can be established. Otherwise, they are not allowed entry.

The next thing was the encryption algorithm itself. The team chose RSA, since we have spent time in class learning and studying how it functioned. The RSA dual-key algorithm focused on a framework where each user has a pair of two keys in order to encrypt and decrypt data. This algorithm is tough to crack mathematically or with brute force since the 2048-bit key is quite large. At this point, all data that is being sent to and from the chat applications are encrypted. The last step was to figure out a way to secure the keys safely and effectively.

5.2 Architecture

This proved to be the most complicated and thought out portion of the project. The team wanted to find a key distribution architecture that would be effective for a group chat, however, secure enough to make sure attackers couldn't get their hands on the keys or important information. The decided upon system features dual-key encryption with a group chat session key. The diagram below shows the architecture that was implemented in Amity Communications v2.0. Shown is the way in which a client joins the chat, and how a message is sent.

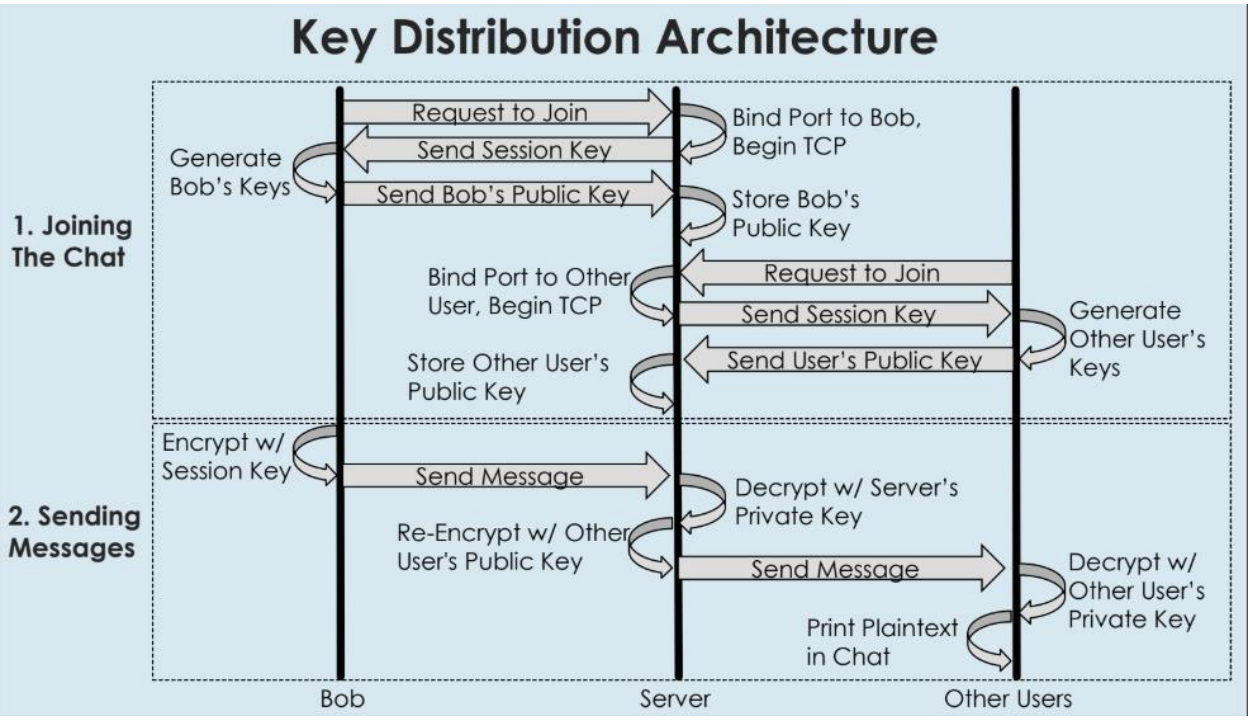

Figure 3: Key Distribution Architecture

Whenever a new client joins the chat, including the first person, the above sequence diagram is followed. Each person has a pair of 2 keys that is generated upon joining the chat, a public key and a private key. The server also follows this rule, having a private key and a session

key (the server public key). As a new person joins and the TCP session is started, the person trades their public key for the session key. Now, the server has all the client's public keys stored, and each client has their private key as well as the session key.

With the keys spread out in this manner, the process of sending messages is quite secure. When a user wants to send a message, they do so by encrypting the data with the session key and sending the message to the server. Once the server receives this message, it decrypts the data with the server's private key, leaving them with the unencrypted message. At this point, since the server has all the client's public keys stored, it re-encrypts the data with each person's public keys, sending a personized message to every client directly connected to the server through TCP. Once the client receives this message, all they must do is decrypt the data using their respective private key. This is effective since all the public keys are used for encryption and all the private keys are used for decryption. That way, even if an attacker got ahold of the encrypted message, and public keys, they still couldn't decrypt the message.

6. Results

Figure 5 shows the GUI that we used for the Amity Communications. It was important that we keep the look of the GUI the same and only changed the architecture of how the server worked. Figure 4 shows print outs from the server that show communication with the clients. The output shows how a client would connect to a server. The first output shows that a client with the initial "CSW" joined the chat room. The client then sends their public key to the server. That message is encrypted with the session key that it received when it connected. Once this is established the client "CSW" then sends a message to the chat room. This is the second output.

You can see both the cyphertext and the plaintext of that message. The server then sends the cyphertext to the clients connected to the chat room. It was also important that all messages sent over the network were encrypted and that the either the client or the sever decrypts the message. This is also proven with the output shown in Figure 4.

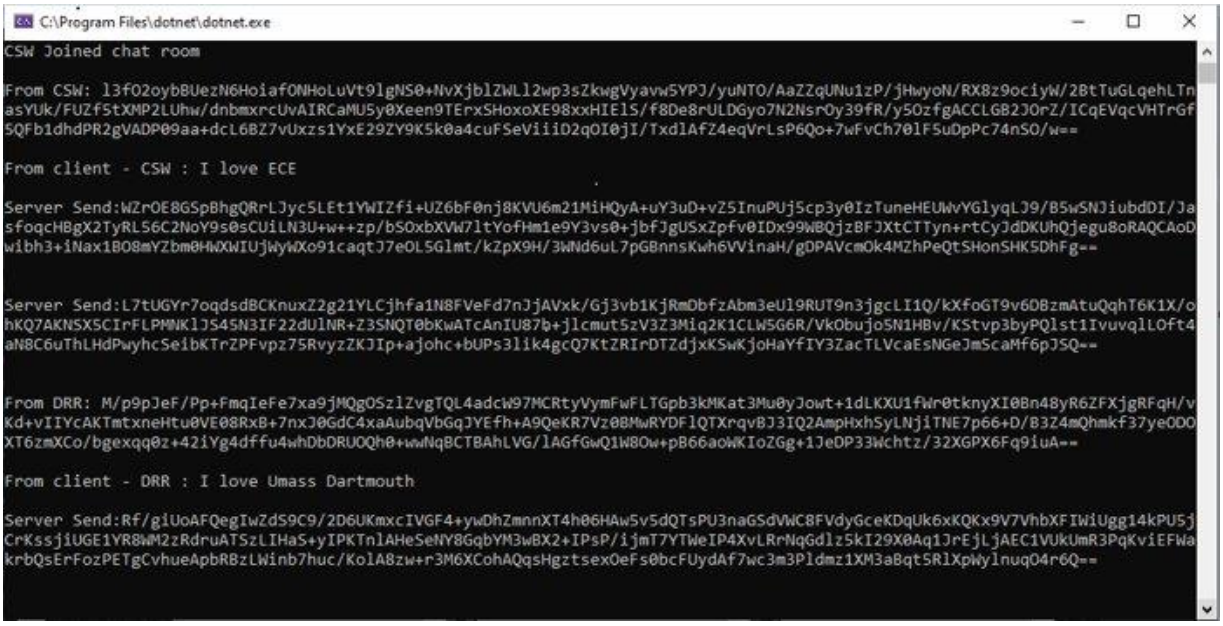

Figure 4: ACv2.0 Server Example

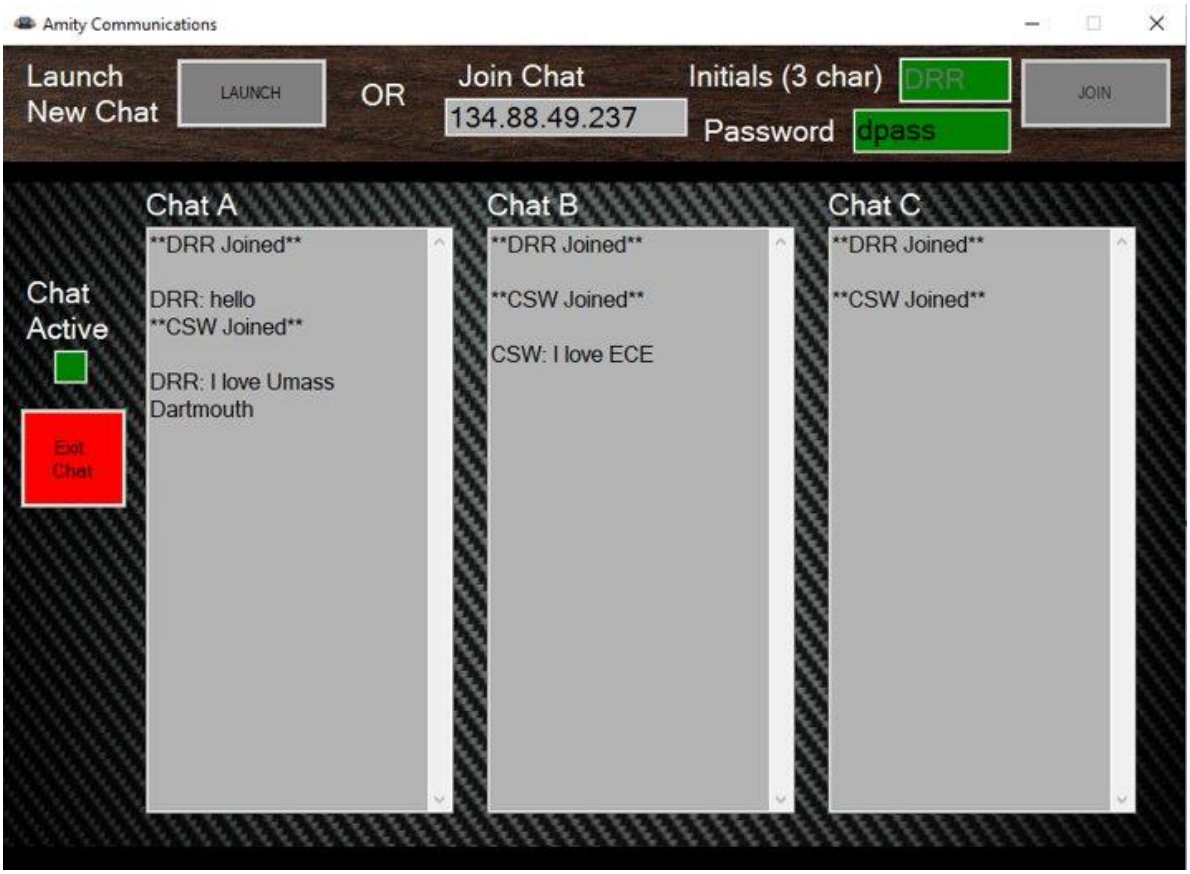

Figure 5: ACv2.0

Also shown at the top right of the UI is the user login area. The initials textbox only accepts initials that are 3-difgits long and that are registered into the system ahead of time. Next, the password must match the stored password for that user in order to establish a legitimate TCP connection to the server and receive the session key.

7. Conclusion

7.1 Lessons Learned

There were many lessons that were learned while working on the project. The first is that team was given a good recap on skills that were learned during ECE 369, such as TCP

connections and group chat messaging. We also learned how valuable encryption is, by seeing how easy it was to steal information from our old chat system. We also learned that there are many ways to add protection to a messaging system, as we cycled through many options when trying to find the best fit for our project. Upon choosing RSA to be the best fit, we gained knowledge on how it can be used on a group chat messaging system. This included learning about dual-key encryption and figuring out a way to distribute all the private keys to each member in the chat. Lastly, the team learned how important data and message security is overall and why it is such a massive concern in today's day-in-age.

7.2 Suggested improvements

Overall, the course was organized and very informative. As seen from section 7.1, many lessons were learned. The take home exam and the project were both a great way to get a hands-on and real-life experience of data protection. If an improvement had to be suggested, one might be that it would be useful to go over code snippets for encryption during lectures. In conclusion, however, the course was nicely structured and little to no improvement is necessary.

8. References

1. "Easy Guide to Encryption and Why It Matters." *Easy Guide to Encryption and Why It Matters | Amnesty International*, https://www.amnesty.org/en/latest/campaigns/2016/10/easy-guide-to-encryption-and-why-it-matters/.

2. Upganlawar, Sooryavanshi, Bhosle, et al. “Digital Signatures for Secure Communication.” *Digital Signatures for Secure Communication - IEEE Conference Publication*, 21 June 2018, https://ieeexplore.ieee.org/document/8389291.

9. Appendix

9.1 Amity Communications v2.0 C# Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Threading;
using System.Net.Sockets;
using System.Net.Security;
using System.Net.NetworkInformation;
using System.Diagnostics;
using System.IO;
using System.Security.Cryptography;
using System.Media;


namespace GUI
{

   public partial class Form1 : Form
   {
      System.Net.Sockets.TcpClient clientSocket = new System.Net.Sockets.TcpClient();
      NetworkStream serverStream = default(NetworkStream);
      string readData = "";

      double d;

      double n;
      public static string IPz;
```

```
    static RSAParameters serverPubKey;

    public Form1()
    {
       InitializeComponent();
    }

    public static class Globals
    {
       public static IPAddress ipAddress = IPAddress.Parse(IPz);
       public static IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());

       public static IPEndPoint remoteEP = new IPEndPoint(ipAddress, 8050);
       public static Socket Sender;

       public static bool running = false;
       public static string data = null;

      public static RSAParameters myPubKey;
      public static RSAParameters myPrivKey;
    }

//*************************************************************************************************
************************
    private void Form1_Load(object sender, EventArgs e)
    {
       ChatCB.SelectedIndex = 0;
       ActiveTB.BackColor = Color.Red;
       ExitB.Hide();
       SendB.Enabled = false;
       InputTB.Enabled = false;
       ChatCB.Enabled = false;


    }
//*************************************************************************************************
************************
    private void JoinB_Click(object sender, EventArgs e)
    {
       string dataFromServer = "";
       if (InitTB.Text.ToUpper() == "CSW")
       {
          if (PassTB.Text != "cpass")
          {
             MessageBox.Show("Password Incorrect.");
             return;
          }
       }
       else if (InitTB.Text.ToUpper() == "PSM")
       {

          if (PassTB.Text != "ppass")
          {
```

```
        MessageBox.Show("Password Incorrect.");
        return;
    }
}

else if (InitTB.Text.ToUpper() == "DRR")
{

    if (PassTB.Text != "dpass")
    {
        MessageBox.Show("Password Incorrect.");
        return;
    }
}

else if (InitTB.Text.ToUpper() == "JJM")
{

    if (PassTB.Text != "jpass")
    {
        MessageBox.Show("Password Incorrect.");
        return;
    }
}
else
{
    MessageBox.Show("User not registered.");
    return;
}
    int NumInit = InitTB.Text.Length;
if (NumInit == 3)
{
    byte[] bytes = new byte[1024];
    string S = "";
    try
    {
        clientSocket.Connect(IPz, 54206);
        serverStream = clientSocket.GetStream();

        string output = "3" + InitTB.Text + "$";
        byte[] outStream = System.Text.Encoding.ASCII.GetBytes(output);
        serverStream.Write(outStream, 0, outStream.Length);

        Thread.Sleep(500);

        serverStream.Read(bytes, 0, bytes.Length);
        dataFromServer = System.Text.Encoding.ASCII.GetString(bytes);
        dataFromServer = dataFromServer.Substring(0, dataFromServer.IndexOf("$"));
        serverStream.Flush();
        //get a stream from the string
        var sr = new System.IO.StringReader(dataFromServer);
        //we need a deserializer
        var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));
```

```
            //get the object back from the stream
            serverPubKey = (RSAParameters)xs.Deserialize(sr);

            var csp = new RSACryptoServiceProvider(2048);

            //how to get the private key
            Globals.myPrivKey = csp.ExportParameters(true);

            //and the public key ...
            Globals.myPubKey = csp.ExportParameters(false);

            string myPubKeyString;
            {
               //we need some buffer
               var sw = new System.IO.StringWriter();
               //we need a serializer
               var x = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));
               //serialize the key into the stream
               x.Serialize(sw, Globals.myPubKey);
               //get the string from the stream
               myPubKeyString = sw.ToString();


            }
            Thread.Sleep(500);

            byte[] Stream = System.Text.Encoding.ASCII.GetBytes(myPubKeyString + "$");
            serverStream.Write(Stream, 0, Stream.Length);

            Thread ctThread = new Thread(getMessage);
            ctThread.Start();
            Globals.running = true;
            JoinB.Enabled = false;
            InitTB.Enabled = false;
            SendB.Enabled = true;
            InputTB.Enabled = true;
            ChatCB.Enabled = true;
            ActiveTB.BackColor = Color.Green;
            ActiveL.Text = "Chat\nActive";
            ExitB.Show();
         }
         catch (Exception ex)
         {
            System.Windows.Forms.MessageBox.Show(ex.ToString());
         }

      }
      else
         System.Windows.Forms.MessageBox.Show("Initials must be\n3 characters");
   }
//*****************************************************************************************
**********************
   private void ChooseChatCLB_ItemCheck(object sender, ItemCheckEventArgs e)
```

```
    {
        for (int ix = 0; ix < ChooseChatCLB.Items.Count; ++ix)
            if (ix != e.Index) ChooseChatCLB.SetItemChecked(ix, false);

    }
//*************************************************************************************************
************************
    private void ExitB_Click(object sender, EventArgs e)
    {

        byte[] outStream = System.Text.Encoding.ASCII.GetBytes(InitTB.Text + "_QUITTING$");
        serverStream.Write(outStream, 0, outStream.Length);
        serverStream.Flush();
        System.Windows.Forms.MessageBox.Show("You have left the chat...");

        ChatATB.Text = String.Empty;
        ChatBTB.Text = String.Empty;
        ChatCTB.Text = String.Empty;
        InputTB.Text = String.Empty;
        InitTB.Text = String.Empty;
        ActiveL.Text = "Chat\nInactive";
        ActiveTB.BackColor = Color.Red;

        JoinB.Enabled = true;
        InitTB.Enabled = true;

        ExitB.Hide();
        clientSocket.GetStream().Close();
    }
//*************************************************************************************************
************************
    private void JoinChatTB_TextChanged(object sender, EventArgs e)
    {

        IPz = JoinChatTB.Text;
    }
//*************************************************************************************************
************************
    private void JoinChatTB_Click(object sender, EventArgs e)
    {
        if (JoinChatTB.Text == "IP Address")
        {
            JoinChatTB.Text = "";
            JoinChatTB.ForeColor = Color.Black;
        }
    }
//*************************************************************************************************
************************
    private void InitTB_TextChanged_1(object sender, EventArgs e)
    {
        int NumInit = InitTB.Text.Length;
        if (NumInit != 3)
            InitTB.BackColor = Color.Red;
```

```
        else
            InitTB.BackColor = Color.Green;
    }
//*****************************************************************************************
************************


    private void SendB_Click_1(object sender, EventArgs e)
    {
        //Encryption


        int NumInit = InitTB.Text.Length;
        if (NumInit == 3)
        {
            try
            {
                string S = "";
                S = ChatCB.SelectedIndex + InitTB.Text + InputTB.Text;

                //    //we have a public key ... let's get a new csp and load that key
                var csp = new RSACryptoServiceProvider();
                csp.ImportParameters(serverPubKey);

                //for encryption, always handle bytes...
                var bytesPlainTextData = System.Text.Encoding.Unicode.GetBytes(S);

                //apply pkcs#1.5 padding and encrypt our data
                var bytesCypherText = csp.Encrypt(bytesPlainTextData, false);

                //we might want a string representation of our cypher text... base64 will do
                var cypherText = Convert.ToBase64String(bytesCypherText);


                //SENDING
                var outStream = System.Text.Encoding.ASCII.GetBytes(cypherText + "$");
                serverStream.Write(outStream, 0, outStream.Length);
                serverStream.Flush();
                InputTB.Text = String.Empty;
            }
            catch (Exception ex)
            {
                System.Windows.Forms.MessageBox.Show(ex.ToString());
            }

        }
        else
            System.Windows.Forms.MessageBox.Show("Initials must be\n3 characters");
    }
//*****************************************************************************************
************************
    private void getMessage()
    {
```

```
        try
        {
           while (true)
           {
              serverStream = clientSocket.GetStream();

              double[] decrypted = new double[1024];
              int buffSize = 0;
              byte[] inStream = new byte[10025];
              buffSize = clientSocket.ReceiveBufferSize;
              serverStream.Read(inStream, 0, inStream.Length);
              string returndata = System.Text.Encoding.ASCII.GetString(inStream);
              returndata = returndata.Substring(0, returndata.LastIndexOf("$"));
              var csp = new RSACryptoServiceProvider();
              csp.ImportParameters(Globals.myPrivKey);
              var bytesCypherText = Convert.FromBase64String(returndata);



              //decrypt and strip pkcs#1.5 padding
              var bytesPlainTextData = csp.Decrypt(bytesCypherText, false);
              readData = System.Text.Encoding.Unicode.GetString(bytesPlainTextData);

              msg();
           }
        }
        catch (Exception ex)
        {
           System.Windows.Forms.MessageBox.Show(ex.ToString());
        }
     }
//*************************************************************************************************
************************
     private void msg()
     {

        string S = ChatCB.SelectedIndex + InitTB.Text + InputTB.Text;
        string MessID = readData.Substring(1, 3);
        string outputText = readData.Substring(4);
        if (this.InvokeRequired)
           this.Invoke(new MethodInvoker(msg));
        else
           switch ((int)Char.GetNumericValue(readData[0]))
           {
              case 0:
                 ChatATB.AppendText(MessID + ": " + outputText);
                 ChatATB.Text += "\r\n";
                 break;
              case 1:
                 ChatBTB.AppendText(MessID + ": " + outputText);
                 ChatBTB.Text += "\r\n";
                 break;
              case 2:
```

```
                ChatCTB.AppendText(MessID + ": " + outputText);
                ChatCTB.Text += "\r\n";
                break;
            case 3:
                readData = readData.Substring(1, readData.Length - 1);
                ChatATB.AppendText(readData + Environment.NewLine);
                ChatBTB.AppendText(readData + Environment.NewLine);
                ChatCTB.AppendText(readData + Environment.NewLine);
                ChatATB.Text += "\r\n";
                ChatBTB.Text += "\r\n";
                ChatCTB.Text += "\r\n";
                break;

        }
    }

    private void LaunchB_Click(object sender, EventArgs e)
    {
        ProcessStartInfo startInfo = new ProcessStartInfo();
        startInfo.FileName = @"D:\FINAL SYSTEM3.0\ServerTest\ServerTest\bin\Debug\netcoreapp2.0\win10-
x64\ServerTest.exe";

        //startInfo.CreateNoWindow = true;
        startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Minimized;
        Process.Start(startInfo);
    }

    private void ChatATB_TextChanged(object sender, EventArgs e)
    {
        ChatATB.SelectionStart = ChatATB.TextLength;
        ChatATB.ScrollToCaret();
    }

    private void ChatBTB_TextChanged(object sender, EventArgs e)
    {
        ChatBTB.SelectionStart = ChatBTB.TextLength;
        ChatBTB.ScrollToCaret();
    }

    private void ChatCTB_TextChanged(object sender, EventArgs e)
    {
        ChatCTB.SelectionStart = ChatCTB.TextLength;
        ChatCTB.ScrollToCaret();
    }

    private void Label8_Click(object sender, EventArgs e)
    {

    }

    private void PassTB_TextChanged(object sender, EventArgs e)
    {
        int NumChar = PassTB.Text.Length;
```

```
        if (NumChar != 0)
            PassTB.BackColor = Color.Green;
        else
            PassTB.BackColor = Color.Red;
    }


//*************************************************************************************************
************************
  }
}
```

9.2 Server C# Code

```
using System;

using System.Threading;

using System.Net.Sockets;

using System.Text;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Security.Cryptography;


namespace ConsoleApplication1

{

  static class Globals

  {

    public static RSAParameters privKey;

  }

  class Program

  {

    public static Hashtable clientsList = new Hashtable();
```

```
public class Person
{
   public int hash;
   public string initials;
   public string publicKey;
}
public static List<Person> personList = new List<Person>();

static void Main(string[] args)
{
   System.Diagnostics.ProcessStartInfo start = new System.Diagnostics.ProcessStartInfo();
   start.FileName = @"\ServerTest.exe";
   start.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;

   TcpListener serverSocket = new TcpListener(54206);
   TcpClient clientSocket = default(TcpClient);
   int counter = 0;

   serverSocket.Start();
   Console.WriteLine("Chat Server Started ....");
   counter = 0;
   while ((true))
   {
      counter += 1;
      clientSocket = serverSocket.AcceptTcpClient();
```

```
byte[] bytesFrom = new byte[10025];

string dataFromClient = null;

string clientPubKey = null;


NetworkStream networkStream = clientSocket.GetStream();

networkStream.Read(bytesFrom, 0, bytesFrom.Length);

dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);

dataFromClient = dataFromClient.Substring(0, dataFromClient.IndexOf("$"));


networkStream.Flush();


//lets take a new CSP with a new 2048 bit rsa key pair

var csp = new RSACryptoServiceProvider(2048);


//how to get the private key

Globals.privKey = csp.ExportParameters(true);


//and the public key ...

var pubKey = csp.ExportParameters(false);


//converting the public key into a string representation

string pubKeyString;

{

   //we need some buffer

   var sw = new System.IO.StringWriter();
```

```
    //we need a serializer
    var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));
    //serialize the key into the stream
    xs.Serialize(sw, pubKey);
    //get the string from the stream
    pubKeyString = sw.ToString();


}
byte[] outStream = System.Text.Encoding.ASCII.GetBytes(pubKeyString + "$");

networkStream.Write(outStream, 0, outStream.Length);

Thread.Sleep(500);


networkStream.Read(bytesFrom, 0, bytesFrom.Length);

clientPubKey = System.Text.Encoding.ASCII.GetString(bytesFrom);
clientPubKey = clientPubKey.Substring(0, clientPubKey.IndexOf("$"));

clientsList.Add(dataFromClient, clientSocket);

var per = new Person();
per.hash = clientSocket.GetHashCode();
per.initials = dataFromClient.Substring(1);
per.publicKey = clientPubKey;
```

```
        personList.Add(per);

        broadcast("3**" + dataFromClient.Substring(1) + " Joined**", dataFromClient, false);

        Console.WriteLine(dataFromClient.Substring(1) + " Joined chat room ");
        handleClinet client = new handleClinet();
        client.startClient(clientSocket, "0"/*dataFromClient.Substring(1)*/, clientsList);



    }

    clientSocket.Close();
    serverSocket.Stop();
    Console.WriteLine("exit");
    Console.ReadLine();
}

public static void broadcast(string msg, string uName, bool flag)
{

    foreach (DictionaryEntry Item in clientsList)
    {
        int i = 0;
        TcpClient broadcastSocket;
        broadcastSocket = (TcpClient)Item.Value;
        NetworkStream broadcastStream = broadcastSocket.GetStream();
```

```
Byte[] broadcastBytes = null;
while (broadcastSocket.GetHashCode() != personList[i].hash)
{
   i++;
}
var sr = new System.IO.StringReader(personList[i].publicKey);

//we need a deserializer
var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));
//get the object back from the stream
RSAParameters PubKey = (RSAParameters)xs.Deserialize(sr);
var csp = new RSACryptoServiceProvider();
csp.ImportParameters(PubKey);

//for encryption, always handle bytes...
var bytesPlainTextData = System.Text.Encoding.Unicode.GetBytes(msg);

//apply pkcs#1.5 padding and encrypt our data
var bytesCypherText = csp.Encrypt(bytesPlainTextData, false);

//we might want a string representation of our cypher text... base64 will do
var cypherText = Convert.ToBase64String(bytesCypherText);
Console.WriteLine("\nServer Send:" + cypherText + "\n");
broadcastBytes = System.Text.Encoding.ASCII.GetBytes(cypherText + "$");

broadcastStream.Write(broadcastBytes, 0, broadcastBytes.Length);
broadcastStream.Flush();
```

```
        }
    }  //end broadcast function
}//end Main class



public class handleClinet
{
    TcpClient clientSocket;
    string clNo;
    Hashtable clientsList;

    public void startClient(TcpClient inClientSocket, string clineNo, Hashtable cList)
    {
        this.clientSocket = inClientSocket;
        this.clNo = clineNo;
        this.clientsList = cList;
        Thread ctThread = new Thread(doChat);
        ctThread.Start();
    }

    private void doChat()
    {
        int requestCount = 0;
        byte[] bytesFrom = new byte[10025];
        string dataFromClient = null;
        Byte[] sendBytes = null;
        string serverResponse = null;
```

```
string rCount = null;
requestCount = 0;

var csp = new RSACryptoServiceProvider();
csp.ImportParameters(Globals.privKey);

while ((true))
{
   try
   {
      requestCount = requestCount + 1;
      NetworkStream networkStream = clientSocket.GetStream();
      networkStream.Read(bytesFrom, 0, bytesFrom.Length);

      dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
      dataFromClient = dataFromClient.Substring(0, dataFromClient.LastIndexOf("$"));


      var bytesCypherText = Convert.FromBase64String(dataFromClient);



      //decrypt and strip pkcs#1.5 padding
      var bytesPlainTextData = csp.Decrypt(bytesCypherText, false);
      var PlainTextData = System.Text.Encoding.Unicode.GetString(bytesPlainTextData);

      clNo = PlainTextData.Substring(1, 3);
```

```
    Console.WriteLine("\nFrom " + clNo + ": " + dataFromClient + "\n");


    //get our original plainText back...
    if (dataFromClient.Substring(3) == "_QUITTING")
    {
        clientsList.Remove(dataFromClient.Substring(0, 3));
        networkStream.Close();
        clientSocket.Close();
        Console.WriteLine("CLIENT LEAVING: " + dataFromClient.Substring(0, 3));
        rCount = Convert.ToString(requestCount);
        Program.broadcast("3**" + dataFromClient.Substring(0, 3) + " has left the chat**", clNo, true);
        return;
    }
    if (dataFromClient.Length == 4)
    {
        //Dont send empty message
    }
    else
    {
        Console.WriteLine("From client - " + clNo + " : " + PlainTextData.Substring(4));
        rCount = Convert.ToString(requestCount);


        Program.broadcast(PlainTextData, clNo, true);
    }
}
catch (Exception ex)
{
```

```
                Console.WriteLine(ex.ToString());
            }
        }//end while
    }//end doChat
} //end class handleClinet
}//end namespace
```