

Encrypted TCP chat using RSA and AES algorithm

Prof.Dr. fatih abut

By

Karrar Ali Mohsion Al-Hameedawi

Student Number: 2023913124

Department: Computer Engineering

Program Name: PhD

# Introduction

The Secure Chat Application is a sophisticated client-server communication tool developed using C#. It is specifically designed to enable real-time chat functionality while ensuring that all communications remain secure. The application leverages socket programming to facilitate seamless data exchange between the client and the server, ensuring low-latency and reliable message delivery.

To protect the confidentiality and integrity of the messages, the application employs a dual-layered encryption approach. Symmetric encryption is implemented using the Advanced Encryption Standard (AES), which is renowned for its efficiency and strong security. AES ensures that the message content is encrypted with a secret key that is only known to the communicating parties.

In addition to AES, the application uses RSA for asymmetric encryption, providing an extra layer of security. RSA encryption allows the secure exchange of keys and ensures that even if the communication channel is compromised, the actual message content remains inaccessible to unauthorized users. By using public and private key pairs, RSA facilitates secure key exchange and authentication processes.

The combination of AES and RSA encryption mechanisms ensures that all data transmitted between the client and server is not only encrypted but also secure from potential eavesdropping or tampering. This dual-encryption strategy makes the Secure Chat Application a robust solution for environments where secure communication is paramount.

Beyond its security features, the application is designed for ease of use and efficient performance. It can handle multiple clients simultaneously, ensuring that all users can communicate in real time without any noticeable delay. The user interface is intuitive, allowing users to easily send and receive messages while the underlying encryption processes occur seamlessly in the background.

Overall, the Secure Chat Application represents a comprehensive solution for secure, real-time communication. By integrating advanced encryption techniques within a user-friendly client-server architecture, it provides both security and usability, making it ideal for any setting that requires confidential and reliable communication.

# The project's aim

The application ought to be, as previously indicated, fast, dependable, able to operate over TCP protocol, able to facilitate user-to-user communication, compatible with Windows 7 x64 computers, and easily adaptable to future changes in the user interface.   
Utilizing cryptographic functions was chosen in order to meet the aforementioned requirements:   
The Advanced Encryption Standard, the RSA algorithm, and the AES algorithm   
The initiative was predicated on allowing communication to occur solely between two people at once. There should be two distinct channels for communication: one for receiving messages and the other for sending them. Each channel functions independently of the others. Additionally, although the program has the ability to generate fresh key pairs, it was expected that users exchanged public keys prior to establishing communication.

# Features

* Real-time chat between multiple clients and a server.
* RSA encryption for secure key exchange.
* AES encryption for secure message transmission.
* GUI interface for both client and server.
* Ability to switch between encrypted and unencrypted message

# System Architecture

The application consists of two main components:

* Server: Handles incoming connections, manages active clients, and relays messages.
* Client: Connects to the server, sends and receives messages.

# Components Interaction

* Connection Setup: The client connects to the server using sockets.
* Encryption Key Exchange:

RSA is used for securely exchanging AES keys.

The server generates a public/private RSA key pair and shares the public key with the client.

The client uses the server’s public key to encrypt an AES key and sends it to the server.

The server decrypts the AES key using its private key.

## Secure Communication:

Once the AES key is exchanged, all messages between the client and server are encrypted using AES.

### Installation and Setup

Prerequisites

.NET Framework

Visual Studio

## Open the Project:

* Open Visual Studio.
* Navigate to the cloned repository and open the solution file.

## Build the Solution:

* Build the solution to restore the NuGet packages and compile the code.

## Run the Server:

* Set the server project (Serverthang) as the startup project.
* Run the server application.

## Run the Client:

* Set the client project (Clientthang) as the startup project.
* Run the client application.

# Components

## Server

The server component is responsible for listening for incoming client connections, managing connected clients, and handling the reception and broadcasting of messages. The server encrypts the messages using RSA before sending them to the clients.

# Key Classes and Methods

## Form1: The main form of the server application.

* Connect(): Establishes the server socket and begins listening for client connections.
* SendRSA(Socket client): Encrypts and sends messages using RSA.
* ReceiveMessage(object obj): Receives messages from clients and broadcasts them to other clients.
* CreateNewKeys(): Generates new RSA public and private keys.
* Encryption(): Encrypts a message using the RSA public key.
* Decryption(): Decrypts a message using the RSA private key.
* RsaEnc: Helper class for handling RSA encryption and decryption.
* Encrypt(string plainText): Encrypts a plaintext message.
* Decrypt(string cypherText): Decrypts an encrypted message.

using System;

using System.Text;

using System.Collections.Generic;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

using System.Security;

using System.Threading.Tasks;

using System.Threading;

using System.Security.Cryptography;

using Microsoft.VisualBasic;

using System.Xml;

namespace Serverthang

{

    public partial class Form1 : Form

    {

        RsaEnc rs = new RsaEnc();

        public Form1()

        {

            InitializeComponent();

            CheckForIllegalCrossThreadCalls = false;

            Connect();

        }

        private void ButtonStartServer\_Click(object sender, EventArgs e)

        {

        }

        private void ButtonStopServer\_Click(object sender, EventArgs e)

        {

        }

        private void ButtonSendRequest\_Click(object sender, EventArgs e)

        {

            if (comboBox1.SelectedItem != "RSA")

            {

                MessageBox.Show("Not encrypted");

                foreach (Socket item in clientList)

                {

                    SendMesseage(item);

                }

                AddMesseage(textBox1.Text);

                textBox1.Clear();

            }

            else if (comboBox1.SelectedItem == "RSA")

            {

                MessageBox.Show("RSA");

                foreach (Socket item in clientList)

                {

                    SendRSA(item);

                }

                AddMesseage(textBox1.Text);

            }

        }

        //Using RsaENC

        /\*

        void SendRSA(Socket client)

        {

            string text = textBox1.Text;

            rs.Encrypt(text);

            string txtsend = rs.Encrypt(text);

            byte[] request = Serialize(txtsend);

            if (request != null)

            {

                client.Send(Serialize(txtsend));

            }

            txtshow.AppendText(text);

            txtshow.AppendText(": ");

            AddMesseage(txtsend);

        }

        \*/

        void SendRSA(Socket client)

        {

            string text = textBox1.Text;

            plaintext = ByteConverter.GetBytes(text);

            string txtsend = Encryption(text);

            byte[] request = Serialize(txtsend);

            if (request != null)

            {

                client.Send(Serialize(txtsend));

                //serverthang.Send(request);

            }

            txtshow.AppendText(text);

            txtshow.AppendText(": ");

            AddMesseage(txtsend);

        }

        //Using RsaENC

        void SendKey1(Socket client)

        {

            //string Key = rs.PublicKeyString();

            byte[] request = Serialize(rs.PublicKeyString());

            if (request != null)

            {

                client.Send(Serialize(rs.PublicKeyString()));

                //serverthang.Send(request);

            }

        }

        void Goitin(Socket clientthang)

        {

            string text = textBox1.Text;

            if (textBox1.Text != null)

            {

                serverthang.Send(Serialize(textBox1.Text));

            }

        }

        void SendMesseage(Socket client)

        {

            //string text = textBox1.Text;

            byte[] request = Serialize(textBox1.Text);

            if (request != null)

            {

                client.Send(Serialize(textBox1.Text));

                //serverthang.Send(request);

            }

        }

        //Using Internal CreateNewKeys()

        void SendKey(Socket client)

        {

            string Key = txtPublickey.Text;

            byte[] request = Serialize(Key);

            if (request != null)

            {

                client.Send(Serialize(Key));

            }

        }

        void ReceiveMesseage(object obj)

        {

            Socket client = obj as Socket;

            try

            {

                while (true)

                {

                    byte[] data = new byte[1024 \* 5000];

                    client.Receive(data);

                    //string messeage = Encoding.UTF8.GetString(data);

                    string messeage = (string)Derserialize(data);

                    foreach (Socket item in clientList)

                    {

                        if (item != null)

                        {

                            item.Send(Serialize(messeage));

                        }

                    }

                    AddMesseage(messeage);

                }

            }

            catch

            {

                clientList.Remove(client);

                client.Close();

            }

        }

        byte[] Serialize(object obj)

        {

            MemoryStream stream = new MemoryStream();

            BinaryFormatter formatter = new BinaryFormatter();

            formatter.Serialize(stream, obj);

            return stream.ToArray();

        }

        object Derserialize(byte[] data)

        {

            MemoryStream stream = new MemoryStream(data);

            BinaryFormatter formatter = new BinaryFormatter();

            return formatter.Deserialize(stream);

            return stream.ToArray();

        }

        IPEndPoint IP;

        Socket serverthang;

        List<Socket> clientList;

        void Connect()  //Giong ham StarServer

        {

            clientList = new List<Socket>();

            IP = new IPEndPoint(IPAddress.Any, 9999);

            serverthang = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

            serverthang.Bind(IP);   //Sever make connect, therefore it's waitting //Server tao ra ket noi cho nen no phai ngoi doi

            Thread Listen = new Thread(() => {

                try

                {

                    while (true)

                    {

                        serverthang.Listen(100); //Waitting 100 in stack //Doi 100 dua trong hang cho, doi 100 Stack

                        Socket client = serverthang.Accept();

                        clientList.Add(client);

                        Thread Nhan = new Thread(ReceiveMesseage);

                        Nhan.IsBackground = true;

                        Nhan.Start(client);

                    }

                }

                catch

                {

                    IP = new IPEndPoint(IPAddress.Any, 9999);

                    serverthang = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.IP);

                }

            });

            Listen.IsBackground = true;

            Listen.Start();

        }

        void AddMesseage(string s)

        {

            //lsvMesseage.Items.Add(new ListViewItem() { Text = s });

            //textBox1.Clear();

            txtshow.AppendText(s + Environment.NewLine);

            textBox1.Clear();

        }

        void AddMesseagePublickey(string s)

        {

            //txtPublickey.AppendText(s + Environment.NewLine);

            txtPublickey.Text = s;

            textBox1.Clear();

        }

        void AddMesseagePrivatekey(string s)

        {

            //txtPrivatekey.AppendText(s + Environment.NewLine);

            txtPrivatekey.Text = s;

            textBox1.Clear();

        }

        private void TextBox1\_TextChanged(object sender, EventArgs e)

        {

        }

        private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

        {

            if (comboBox1.SelectedItem == "RSA")

            {

                //MessageBox.Show("The selected plan is RSA");

                label1.Visible =true;

                label3.Visible = true;

                txtPublickey.Visible = true;

                txtPrivatekey.Visible = true;

                BtPublickey.Visible = true;

                btDecrypt.Visible = true;

            }

        }

        private void Form1\_Load(object sender, EventArgs e)

        {

        }

        /\* //Code from Microsoft

        UnicodeEncoding ByteConverter = new UnicodeEncoding();

        RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

        byte[] plaintext;

        byte[] encryptedtext;

        static public byte[] RSAEncrypt(byte[] DataToEncrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

        {

            try

            {

                RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider(2048);

                RSAalg.ImportParameters(RSAKeyInfo);

                return RSAalg.Encrypt(DataToEncrypt, DoOAEPPadding);

            }

            catch (CryptographicException e)

            {

                return null;

            }

        }

        \*/

        /\* //Code from Microsoft

        static public byte[] RSADecrypt(byte[] DataToDecrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

        {

            try

            {

                RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider(2048);

                RSAalg.ImportParameters(RSAKeyInfo);

                return RSAalg.Decrypt(DataToDecrypt, DoOAEPPadding);

            }

            catch (CryptographicException e)

            {

                return null;

            }

        }

        \*/

        UnicodeEncoding ByteConverter = new UnicodeEncoding();

        RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

        byte[] plaintext;

        byte[] encryptedtext;

        public string Encryption(string strText)

        {

            //Import Key from Server

            var publicKey = GetPublicKey();

            var testData = Encoding.UTF8.GetBytes(strText);

            using (var rsa = new RSACryptoServiceProvider(1024))

            {

                try

                {

                    // client encrypting data with public key issued by server

                    rsa.FromXmlString(publicKey.ToString());

                    var encryptedData = rsa.Encrypt(testData, true);

                    var base64Encrypted = Convert.ToBase64String(encryptedData);

                    return base64Encrypted;

                }

                finally

                {

                    rsa.PersistKeyInCsp = false;

                }

            }

        }

        public string Decryption(string strText)

        {

            var privateKey = UsingPrivateKey();

            var testData = Encoding.UTF8.GetBytes(strText);

            using (var rsa = new RSACryptoServiceProvider(1024))

            {

                try

                {

                    var base64Encrypted = strText;

                    // server decrypting data with private key

                    rsa.FromXmlString(privateKey);

                    var resultBytes = Convert.FromBase64String(base64Encrypted);

                    var decryptedBytes = rsa.Decrypt(resultBytes, true);

                    var decryptedData = Encoding.UTF8.GetString(decryptedBytes);

                    return decryptedData.ToString();

                }

                finally

                {

                    rsa.PersistKeyInCsp = false;

                }

            }

        }

        private string GetPublicKey()

        {

            string KeyfromClient = txtshow.Text;

            return KeyfromClient;

        }

        private string UsingPrivateKey()

        {

            string ServerPrivateKey = txtPrivatekey.Text;

            return ServerPrivateKey;

        }

        ///\*

        private void CreateNewKeys()

        {

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

            RSACryptoServiceProvider csp = new RSACryptoServiceProvider(2048);

            //how to get the private key

            RSAParameters privKey = csp.ExportParameters(true);

            //and the public key ...

            RSAParameters pubKey = csp.ExportParameters(false);

            //converting the public key into a string representation

            string pubKeyString;

            {

                //we need some buffer

                var sw = new 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();                   //right

                AddMesseagePublickey(pubKeyString);             //right

                //return pubKeyString;

            }

            string privKeyString;

            {

                //we need some buffer

                var sw = new StringWriter();

                //we need a serializer

                var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));

                //serialize the key into the stream

                xs.Serialize(sw, privKey);

                //get the string from the stream

                privKeyString = sw.ToString();

                AddMesseagePrivatekey(privKeyString);

            }

        }

        //\*/

        private void BtPublickey\_Click(object sender, EventArgs e)

        {

            CreateNewKeys();

            foreach (Socket item in clientList)

            {

                SendKey(item);

            }

        }

        private void BtClear\_Click(object sender, EventArgs e)

        {

            txtshow.Clear();

            txtPublickey.Clear();

            txtPrivatekey.Clear();

        }

        private void btDecrypt\_Click(object sender, EventArgs e)

        {

            string output = Decryption(txtshow.Text);

            MessageBox.Show("The result is: " + output);

            txtshow.Clear();

        }

    }

    //Using for try another way. Not using in this Form Доан Конг Тханг

    public class RsaEnc

    {

        private static RSACryptoServiceProvider csp = new RSACryptoServiceProvider(2048);

        RSAParameters \_PrivateKey;

        RSAParameters \_PublicKey;

        public RsaEnc()

        {

            \_PrivateKey = csp.ExportParameters(true);

            \_PublicKey = csp.ExportParameters(false);

        }

        public string PublicKeyString()

        {

            var sw = new StringWriter();

            var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));

            xs.Serialize(sw, \_PublicKey);

            return sw.ToString();

        }

        public string Encrypt(string plainText)

        {

            RSACryptoServiceProvider csp = new RSACryptoServiceProvider();

            csp.ImportParameters(\_PrivateKey);

            var data = Encoding.Unicode.GetBytes(plainText);

            var cypher = csp.Encrypt(data, false);

            return Convert.ToBase64String(cypher);

        }

        public string Decrypt(string cypherText)

        {

            var dataBytes = Convert.FromBase64String(cypherText);

            csp.ImportParameters(\_PrivateKey);

            var plainnext = csp.Decrypt(dataBytes, false);

            return Encoding.Unicode.GetString(plainnext);

        }

    }

}

## Client

The client component connects to the server, sends messages, and receives broadcasts from the server. It also handles encryption and decryption of messages using RSA.

### Key Classes and Methods

* Form1: The main form of the client application.
* Connect(): Establishes the connection to the server.
* SendRSA(): Encrypts and sends messages using RSA.
* ReceiveMessage(): Receives messages from the server.
* CreateNewKeys(): Generates new RSA public and private keys.
* Encryption(): Encrypts a message using the RSA public key.
* Decryption(): Decrypts a message using the RSA private key.
* RsaEnc: Helper class for handling RSA encryption and decryption.
* Encrypt(string plainText): Encrypts a plaintext message.
* Decrypt(string cypherText): Decrypts an encrypted message.

using System;

using System.Text;

using System.Collections.Generic;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows.Forms;

using System.Security;

using System.Threading.Tasks;

using System.Threading;

using System.Security.Cryptography;

using Microsoft.VisualBasic;

using System.Xml;

namespace Clientthang

{

    public partial class Form1 : Form

    {

        RsaEnc rs = new RsaEnc();

        public Form1()

        {

            InitializeComponent();

            CheckForIllegalCrossThreadCalls = false;

            Connect(); // Ket noi ngay khi bat dau, khong dung button connect

        }

        void SendMesseage()

        {

            string text = textBoxRequest.Text;

            byte[] request = Serialize(textBoxRequest.Text);

            if (request != null)

            {

                clientthang.Send(request);

            }

        }

        void SendKey() //Send Public Key

        {

            string Key = txtPublickey.Text;

            byte[] request = Serialize(Key);

            if (request != null)

            {

                clientthang.Send(Serialize(Key));

            }

        }

        void Goitin1()

        {

            string text = textBoxRequest.Text;

            if (textBoxRequest.Text != null)

            {

                clientthang.Send(Serialize(textBoxRequest.Text));

            }

        }

        void SendRSA()

        {

            string text = textBoxRequest.Text;

            plaintext = ByteConverter.GetBytes(text);

            string txtsend = Encryption(text);

            byte[] request = Serialize(txtsend);

            if (request != null)

            {

                clientthang.Send(Serialize(txtsend));

            }

            txtboxshow.AppendText(text);

            txtboxshow.AppendText(": ");

            AddMesseage(txtsend);

        }

        //Using RsaENC

        /\*

        void SendRSA()

        {

            string text = textBoxRequest.Text;

            string txtsend = rs.Encrypt(text);

            byte[] request = Serialize(txtsend);

            if (request != null)

            {

                clientthang.Send(Serialize(txtsend));

            }

            txtboxshow.AppendText(text);

            txtboxshow.AppendText(": ");

            AddMesseage(txtsend);

        }

        \*/

        void Nhantin()

        {

            try

            {

                while (true)

                {

                    byte[] data = new byte[1024 \* 5000];

                    clientthang.Receive(data);

                    string messeage = (string)Derserialize(data);

                    AddMesseage(messeage); // nhận tin từ Server về, chuyển thành 1 chuỗi và add chuỗi vào trong  tin nhắn

                }

            }

            catch

            {

                Close();

            }

        }

        byte[] Serialize(object obj)

        {

            MemoryStream stream = new MemoryStream();

            BinaryFormatter formatter = new BinaryFormatter();

            formatter.Serialize(stream, obj);

            return stream.ToArray();

        }

        object Derserialize(byte[] data)

        {

            MemoryStream stream = new MemoryStream(data);

            BinaryFormatter formatter = new BinaryFormatter();

            return formatter.Deserialize(stream);

            return stream.ToArray();

        }

        IPEndPoint IP;

        Socket clientthang;

        void Connect()

        {

            //IP: Địa chỉ của Server

            IP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9999);

            clientthang = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

            try

            {

                clientthang.Connect(IP);

            }

            catch

            {

                MessageBox.Show("Cannot connect to server", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

                return;

            }

            Thread listen = new Thread(Nhantin);

            listen.IsBackground = true;

            listen.Start();

        }

        void AddMesseage(string s)

        {

            //txtboxshow.AppendText(s + Environment.NewLine);

            txtboxshow.AppendText(s);

            textBoxRequest.Clear();

        }

        private void TextBoxRequest\_TextChanged(object sender, EventArgs e)

        {

        }

        private void ButtonConnect\_Click(object sender, EventArgs e)

        {

            Connect();

        }

        private void ButtonDisconnect\_Click(object sender, EventArgs e)

        {

            clientthang.Disconnect(true);

        }

        private void lvsMesseage\_SelectedIndexChanged(object sender, EventArgs e)

        {

        }

        UnicodeEncoding ByteConverter = new UnicodeEncoding();

        RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

        byte[] plaintext;

        byte[] encryptedtext;

        /\* //Ham Coppy tu Microsoft

        public static byte[] Encryption(byte[] DataToEncrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

        {

            try

            {

                byte[] encryptedData;

                using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

                {

                    RSA.ImportParameters(RSAKeyInfo);

                    string str = RSA.ToXmlString(false);

                    string abc = RSA.KeyExchangeAlgorithm;

                    encryptedData = RSA.Encrypt(DataToEncrypt, DoOAEPPadding);

                }

                return encryptedData;

            }

            catch (CryptographicException e)

            {

                return null;

            }

        }

        \*/

        //Ham tu viet

        public string Encryption(string strText)

        {

            //Import Key from Server

            var publicKey = GetPublicKey();

            var testData = Encoding.UTF8.GetBytes(strText);

            using (var rsa = new RSACryptoServiceProvider(1024))

            {

                try

                {

                    // client encrypting data with public key issued by server

                    rsa.FromXmlString(publicKey.ToString());

                    var encryptedData = rsa.Encrypt(testData, true);

                    var base64Encrypted = Convert.ToBase64String(encryptedData);

                    return base64Encrypted;

                }

                finally

                {

                    rsa.PersistKeyInCsp = false;

                }

            }

        }

        public string Decryption(string strText)

        {

            var privateKey = UsingPrivateKey();

            var testData = Encoding.UTF8.GetBytes(strText);

            using (var rsa = new RSACryptoServiceProvider(1024))

            {

                try

                {

                    var base64Encrypted = strText;

                    // server decrypting data with private key

                    rsa.FromXmlString(privateKey);

                    var resultBytes = Convert.FromBase64String(base64Encrypted);

                    var decryptedBytes = rsa.Decrypt(resultBytes, true);

                    var decryptedData = Encoding.UTF8.GetString(decryptedBytes);

                    return decryptedData.ToString();

                }

                finally

                {

                    rsa.PersistKeyInCsp = false;

                }

            }

        }

        private string UsingPrivateKey()

        {

            string ClientPrivateKey = txtPrivatekey.Text;

            return ClientPrivateKey;

        }

        private void CreateNewKeys()

        {

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

            RSACryptoServiceProvider csp = new RSACryptoServiceProvider(2048);

            //how to get the private key

            RSAParameters privKey = csp.ExportParameters(true);

            //and the public key ...

            RSAParameters pubKey = csp.ExportParameters(false);

            //converting the public key into a string representation

            string pubKeyString;

            {

                //we need some buffer

                var sw = new 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();                   //right

                AddMesseagePublickey(pubKeyString);             //right

                //return pubKeyString;

            }

            string privKeyString;

            {

                //we need some buffer

                var sw = new StringWriter();

                //we need a serializer

                var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));

                //serialize the key into the stream

                xs.Serialize(sw, privKey);

                //get the string from the stream

                privKeyString = sw.ToString();

                AddMesseagePrivatekey(privKeyString);

            }

        }

        private string GetPublicKey()

        {

            string KeyfromServer = txtboxshow.Text;

            return KeyfromServer;

        }

        void AddMesseagePublickey(string s)

        {

            //txtPublickey.AppendText(s + Environment.NewLine);

            txtPublickey.Text = s;

            textBoxRequest.Clear();

        }

        void AddMesseagePrivatekey(string s)

        {

            //txtPrivatekey.AppendText(s + Environment.NewLine);

            txtPrivatekey.Text = s;

            textBoxRequest.Clear();

        }

        private void BtClear\_Click(object sender, EventArgs e)

        {

            txtboxshow.Clear();

            //txtPublickey.Clear();

            //txtPrivatekey.Clear();

        }

        private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

        {

            if (comboBox1.SelectedItem == "RSA")

            {

                //MessageBox.Show("The selected plan is RSA");

                label1.Visible = true;

                label3.Visible = true;

                txtPublickey.Visible = true;

                txtPrivatekey.Visible = true;

                BtPublickey.Visible = true;

                btDecrypt.Visible = true;

            }

        }

        private void BtPublickey\_Click(object sender, EventArgs e)

        {

            CreateNewKeys();

            SendKey();

        }

        private void Clear()

        {

            txtboxshow.Clear();

        }

        private void ButtonSendRequest\_Click(object sender, EventArgs e)

        {

            if (comboBox1.SelectedItem != "RSA")

            {

                MessageBox.Show("NOT Encrypt");

                SendMesseage();

                textBoxRequest.Clear();

            }

            else if (comboBox1.SelectedItem == "RSA")

            {

                MessageBox.Show("RSA");

                {

                    SendRSA();

                }

                AddMesseage(textBoxRequest.Text);

                txtboxshow.Clear();

            }

            }

        private void btDecrypt\_Click(object sender, EventArgs e)

        {

            string output = Decryption(txtboxshow.Text);

            MessageBox.Show("The result is: " + output);

            txtboxshow.Clear();

        }

    }

    //Using for try another way. Not for this Form. Доан Конг тханг

    public class RsaEnc

    {

        private static RSACryptoServiceProvider csp = new RSACryptoServiceProvider(2048);

        RSAParameters \_PrivateKey;

        RSAParameters \_PublicKey;

        public RsaEnc()

        {

            \_PrivateKey = csp.ExportParameters(true);

            \_PublicKey = csp.ExportParameters(false);

        }

        public string PublicKeyString()

        {

            var sw = new StringWriter();

            var xs = new System.Xml.Serialization.XmlSerializer(typeof(RSAParameters));

            xs.Serialize(sw, \_PublicKey);

            return sw.ToString();

        }

        public string Encrypt(string plainText)

        {

            RSACryptoServiceProvider csp = new RSACryptoServiceProvider();

            csp.ImportParameters(\_PrivateKey);

            var data = Encoding.Unicode.GetBytes(plainText);

            var cypher = csp.Encrypt(data, false);

            return Convert.ToBase64String(cypher);

        }

        public string Decrypt(string cypherText)

        {

            var dataBytes = Convert.FromBase64String(cypherText);

            csp.ImportParameters(\_PrivateKey);

            var plainnext = csp.Decrypt(dataBytes, false);

            return Encoding.Unicode.GetString(plainnext);

        }

    }

}

# Encryption and Decryption

## RSA Encryption

RSA (Rivest–Shamir–Adleman) is an asymmetric encryption algorithm widely used for secure data transmission. In RSA, a pair of keys is used: a public key for encryption and a private key for decryption.

## Key Generation

The CreateNewKeys method generates a new RSA key pair. The public key is used to encrypt messages, and the private key is used to decrypt them.

private void CreateNewKeys()

{

    RSACryptoServiceProvider csp = new RSACryptoServiceProvider(2048);

    RSAParameters privKey = csp.ExportParameters(true);

    RSAParameters pubKey = csp.ExportParameters(false);

    var sw = new StringWriter();

    var xs = new XmlSerializer(typeof(RSAParameters));

    xs.Serialize(sw, pubKey);

    string pubKeyString = sw.ToString();

    sw = new StringWriter();

    xs = new XmlSerializer(typeof(RSAParameters));

    xs.Serialize(sw, privKey);

    string privKeyString = sw.ToString();

}

## AES Encryption

AES (Advanced Encryption Standard) is a symmetric encryption algorithm used for secure data transmission. In AES, the same key is used for both encryption and decryption. This method is used for encrypting the actual message content after the secure key exchange.

# Usage

## Server Application

* Start Server: Click the "Start Server" button to start listening for incoming client connections.
* Stop Server: Click the "Stop Server" button to stop the server.
* Send Messages: Enter a message in the text box and click "Send" to broadcast it to all connected clients.
* Generate RSA Keys: Click the "Generate RSA Keys" button to create new RSA keys for encryption.
* View Messages: All received and sent messages will appear in the message display area.

## Client Application

* Connect to Server: The client automatically attempts to connect to the server on startup.
* Disconnect from Server: Click the "Disconnect" button to disconnect from the server.
* Send Messages: Enter a message in the text box and click "Send" to send it to the server.
* RSA Encryption: Select "RSA" from the combo box to enable RSA encryption for messages.
* Decrypt Messages: Click the "Decrypt" button to decrypt received messages using the private key.
* View Messages: All received and sent messages will appear in the message display area.

## Security Considerations

### Encryption Mechanisms

RSA (Rivest–Shamir–Adleman):

### Used for secure key exchange.

Asymmetric encryption with a public/private key pair.

AES (Advanced Encryption Standard):

### Used for encrypting the actual messages.

Symmetric encryption for fast and secure communication.

Key Management

RSA Key Generation:

### The server generates a new RSA key pair upon starting.

The public key is shared with clients to encrypt the AES key.

AES Key Exchange:

The client generates an AES key and encrypts it with the server's public RSA key.

The encrypted AES key is sent to the server, which decrypts it using the private RSA key.

Message Encryption Flow

Client-Side:

Encrypt messages using the shared AES key.

Send encrypted messages to the server.

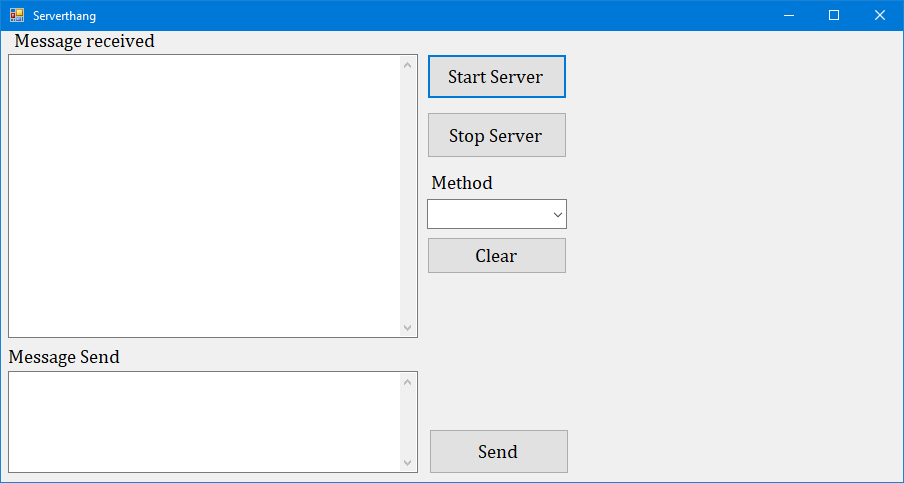
Server-Side:

Decrypt received messages using the shared AES key.

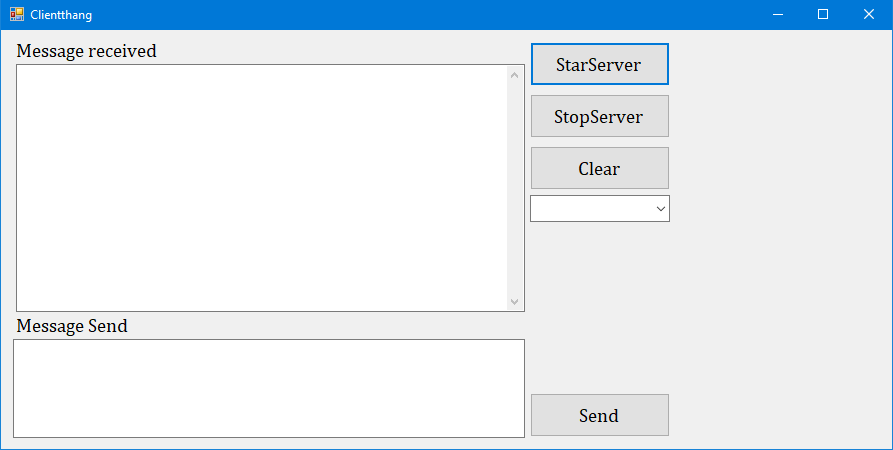
Relay decrypted messages to other clients.

## Run the Chat

1. Open Debug and run serverthang



1. Open Debug and run clientthang



## Code with python language

### Cerver

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

import base64

import os

def aes\_generate\_key():

    # 256bit random data

    return os.urandom(32)

def aes\_gcm\_encrypt(key: bytes, message: bytes) -> bytes:

    # Generate 96bit Initialization Vector

    iv = os.urandom(12)

    # Init AES-GCM

    cipher = Cipher(algorithms.AES(key), modes.GCM(iv))

    encryptor = cipher.encryptor()

    # Encrypt 'message'

    aes\_data = encryptor.update(message) + encryptor.finalize()

    # Get the full cipherText: IV + Encrypted-Message + Auth-Tag

    ciphertext = iv + aes\_data + encryptor.tag

    return ciphertext

def aes\_gcm\_decrypt(key: bytes, ciphertext: bytes) -> bytes:

    iv = ciphertext[0:12]

    aes\_data = ciphertext[12:-16]

    tag = ciphertext[-16:]

    cipher = Cipher(algorithms.AES(key), modes.GCM(iv, tag))

    decryptor = cipher.decryptor()

    text = decryptor.update(aes\_data) + decryptor.finalize()

    return text

if \_\_name\_\_ == '\_\_main\_\_':

    MESSAGE = b'hello world'

    aes\_key = aes\_generate\_key()

    ciphertext = aes\_gcm\_encrypt(aes\_key, MESSAGE)

    plaintext = aes\_gcm\_decrypt(aes\_key, ciphertext)

    assert plaintext == MESSAGE, (plaintext, MESSAGE)

    print('AES key', base64.b64encode(aes\_key))

    print('Ciphertext', base64.b64encode(ciphertext))

**output**

**AES key b'dMWxxh1KqCSLRhU93XPuj5md8LWWtqLTREO8Z45KMbs='**

**Ciphertext b'M/zUOygaL2LEdkZvFH7LtNWSc8Yw03Mnu0GcUBLuyuRW8xxhaUiK'**

## Client

#!/usr/bin/env python3

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

# Copyright (c) 2023 Matteo Bertozzi

#

# Permission is hereby granted, free of charge, to any person obtaining a copy

# of this software and associated documentation files (the "Software"), to deal

# in the Software without restriction, including without limitation the rights

# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

# copies of the Software, and to permit persons to whom the Software is

# furnished to do so, subject to the following conditions:

#

# The above copyright notice and this permission notice shall be included in all

# copies or substantial portions of the Software.

#

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

# SOFTWARE.

#

# pip3 install cryptography

from cryptography.hazmat.primitives.serialization import \*

from cryptography.hazmat.primitives.asymmetric import padding, rsa

from cryptography.hazmat.primitives import hashes

import base64

import os

def generate\_rsa\_keys(key\_size: int = 4096) -> tuple[rsa.RSAPrivateKey, rsa.RSAPublicKey]:

    private\_key = rsa.generate\_private\_key(

        public\_exponent=65537, key\_size=key\_size)

    public\_key = private\_key.public\_key()

    return private\_key, public\_key

def export\_rsa\_public\_key(public\_key: rsa.RSAPublicKey) -> bytes:

    public\_key\_spki = public\_key.public\_bytes(

        encoding=Encoding.DER,

        format=PublicFormat.SubjectPublicKeyInfo

    )

    return public\_key\_spki

def export\_rsa\_private\_key(private\_key: rsa.RSAPrivateKey) -> bytes:

    private\_key\_pkcs8 = private\_key.private\_bytes(

        encoding=Encoding.DER,

        format=PrivateFormat.PKCS8,

        encryption\_algorithm=NoEncryption()  # Optional encryption algorithm

    )

    return private\_key\_pkcs8

def load\_rsa\_public\_key(public\_key\_spki: bytes) -> rsa.RSAPublicKey:

    return load\_der\_public\_key(public\_key\_spki)

def load\_rsa\_private\_key(private\_key\_pkcs8: bytes) -> rsa.RSAPrivateKey:

    return load\_der\_private\_key(private\_key\_pkcs8, password=None)

OAEP\_PADDING = padding.OAEP(

    mgf=padding.MGF1(algorithm=hashes.SHA512()),

    algorithm=hashes.SHA512(),

    label=None

)

def rsa\_oaep\_encrypt(public\_key: rsa.RSAPublicKey, message: bytes) -> bytes:

    return public\_key.encrypt(message, OAEP\_PADDING)

def rsa\_oaep\_decrypt(private\_key: rsa.RSAPrivateKey, ciphertext: bytes) -> bytes:

    return private\_key.decrypt(ciphertext, OAEP\_PADDING)

def rsa\_sign\_sha256(private\_key: rsa.RSAPrivateKey, message: bytes) -> bytes:

    return private\_key.sign(

        message,

        padding.PKCS1v15(),

        hashes.SHA256()

    )

def rsa\_verify\_sign\_sha256(public\_key: rsa.RSAPublicKey, signature: bytes, message: bytes) -> bytes:

    public\_key.verify(

        signature,

        message,

        padding.PKCS1v15(),

        hashes.SHA256()

    )

if \_\_name\_\_ == '\_\_main\_\_':

    MESSAGE = b'hello world'

    # Generate RSA key pair

    private\_key, public\_key = generate\_rsa\_keys()

    # Encrypt and decrypt using RSA-OAEP

    ciphertext = rsa\_oaep\_encrypt(public\_key, MESSAGE)

    plaintext = rsa\_oaep\_decrypt(private\_key, ciphertext)

    # Ensure decryption is successful

    assert plaintext == MESSAGE, (plaintext, MESSAGE)

    # Sign the message and verify the signature

    signature = rsa\_sign\_sha256(private\_key, MESSAGE)

    rsa\_verify\_sign\_sha256(public\_key, signature, MESSAGE)

    # Print results

    print('RSA public-key:', base64.b64encode(export\_rsa\_public\_key(public\_key)).decode('utf-8'))

    print('RSA private-key:', base64.b64encode(export\_rsa\_private\_key(private\_key)).decode('utf-8'))

    print('Ciphertext (Length: {}):'.format(len(ciphertext)), base64.b64encode(ciphertext).decode('utf-8'))

    print('Signature (Length: {}):'.format(len(signature)), base64.b64encode(signature).decode('utf-8'))

**output**

**RSA public-key: **

**RSA private-key: **

**Ciphertext (Length: 512): **

**Signature (Length: 512): **

# Conclusion

This secure chat application demonstrates the integration of socket programming with robust encryption mechanisms to ensure secure communication. By using a combination of RSA for key exchange and AES for message encryption, the application provides a secure environment for real-time chat. The user-friendly interface makes it easy to use, while the underlying encryption ensures that messages remain confidential and tamper-proof during transmission.

For more detailed information on each component, please refer to the respective documentation in the repository. If you encounter any issues or have questions, feel free to open an issue on GitHub or contact the project maintainers.