Client Server Chat Application

M. A. Mioc

University Stefan cel Mare Suceava
Integrated Center for research, development and innovation in
Advanced Materials, Nanotechnologies, and Distributed Systems – MANSiD
Suceava, Romania
mmioc@cs.upt.ro

***Abstract*—Nowadays it is a continuous increasing of the networks interconnections which generates a growing necessity to keep the information secure from the eavesdroppers attacks and also from the hackers. Despite the fact that the scientific fundamentals of Coding Theory are already known for years it is important to present some researches from this area. The development of Coding Theory produced successful applications in Cryptography, in Error Detecting and Correcting Codes and in Wireless Communication Systems. This paper describes a research based on developing a client server chat application. The originality of this work is given by the encryption part of this implementation. So, the encryption part through the encryption password logon, encryption and sending messages keeps them safe from client to client.**

Keywords—cryptography; security; ciphers; plaintext; algorithm; encryption; decryption.

I. INTRODUCTION

In cryptography, RSA is a cryptographic algorithm with public keys, first algorithm used for encrypting and also for electrical signature. Algorithm was developed in 1977 and published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman at MIT having the name composed from all three authors initials [2].

In 1978, in the Communications of Association for Computing Machinery (ACM) some specific methods for obtaining Digital Signatures and Public-Key Cryptosystems were presented by the same team [2]. RSA is a block encrypting algorithm. This means that the initial text (clear text), but also an encrypted text are numbers between 0 and -1, with a chosen n. A message longer than log. n is split in segments of the corresponding length, named blocks, which are encrypted one by one. Also, as cryptographic algorithm with public keys, it functions based on a pair of keys mathematically connected to each other: a public key, well known to all involved parts, and a secret key, known only by its keeper. RSA is used only at the beginning of communication, for transmitting the secret communication key, which is then used in a secret key algorithm, like 3DES or AES.

AES (Advanced Encryption Standard), known also by the name of Rijndael, is a standard algorithm for blocks symmetric encrypting, used today on a large domain of applications and adopted as standard by the Governmental American Association NIST. The new standard algorithm developed by the two Belgian cryptographs, Joan Daemen and Vincent Rijmen became the officially AES and newt to NIST for selection under the name of Rijndael [7].

In the advanced proposal NIST, the two authors of the algorithm Rijndael have defined a block encrypting algorithm having the independent lengths for the block and for the key of 128 bits, 192 bits, or 256 bits. AES specification standardizes all 3 dimensions possible for the length of the key, but restricts the block length at 128 bits. Input and output of encrypting and decrypting algorithm is a block of 128 bits. In FIPS number 197, AES operations are defined under the form of matrix operations, where the key and the block are written in matrix format. At the beginning of cipher run, the block is copied in a table named state having first 4 bites on the first column, then the next 4 on the second continuing in the same way till all table is completed.

II. APPLICATION DESCRIPTION

The application is composed of a server that connects all users logged in the application. In order to begin a chat the program must be ran and the server should be started. In developing this research the main principles from the Handbook Applied Cryptography were respected [3]. Some other specific researches are done in many other domains as military [4], medicine [5] and commerce as well. Another problem is how to choose the most convenient algorithm for a specific application [6]. For this goal a comparison of the most frequently used Encryption Algorithms is important to be taken into account [9].

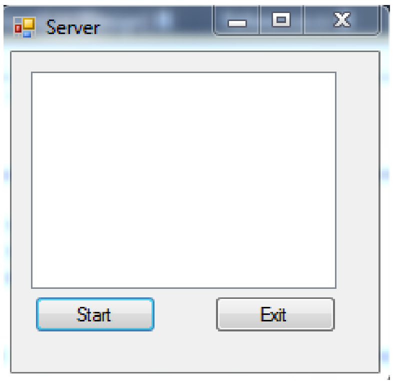
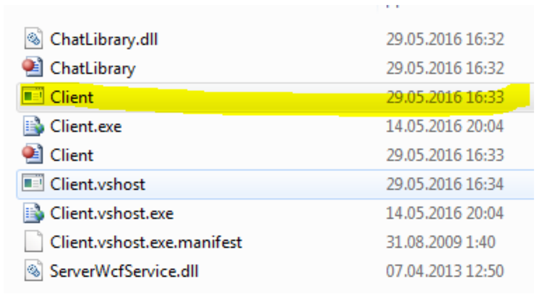


Fig. 1 Server start

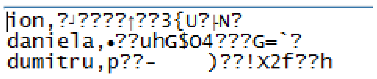
Once the server is started, we can introduce the users in the application, by opening 1, 2 or multiple users. For this we search in project folder and we chose the path "…\ChatServer\Client\bin\Debug" and there we start the executable Client.



ChatLibrary.dll	29.05.2016 16:32
ChatLibrary	29.05.2016 16:32
Client	29.05.2016 16:33
Client.exe	14.05.2016 20:04
Client	29.05.2016 16:33
Client.vshost	29.05.2016 16:34
Client.vshost.exe	14.05.2016 20:04
Client.vshost.exe.manifest	31.08.2009 1:40
ServerWcfService.dll	07.04.2013 12:50

Fig. 2 Executable Client

The user must log in to the application with a UserName and an existing Password from the application. If the Password does not exist in the application, an error message will be shown and all messaging fields will become inactive. For logging in, a text file is created, where the names of all registered users are defined (name:daniela ; pswd: daniela; Name: ion; Pswd: ionel). UserName and password of existing users are saved in a text document from ChatServer "name_passwd.txt" separated by comma, password being under hashed format.



```
ion,?┘????↑??3{U?├N?
daniela,•??uhG$O4???G=`?
dumitru,p??-     )??!x2f??h
```

Fig. 3 UserName and Password part 1 – name_passwd.txt

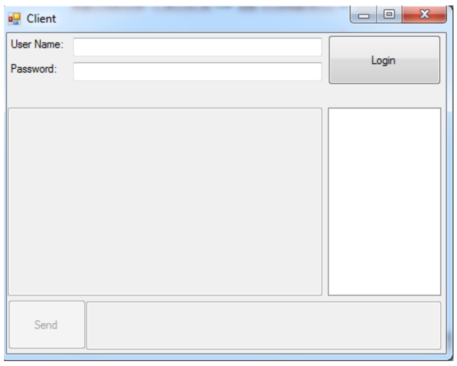
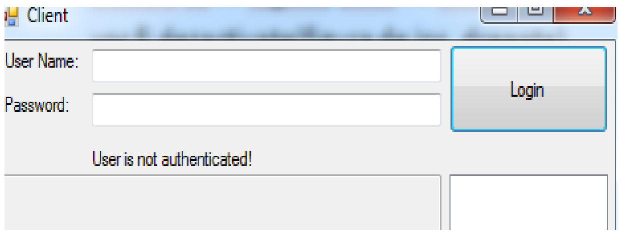


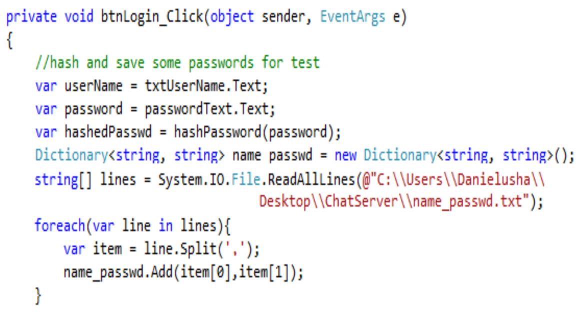
Fig. 4 UserName and Password part 2 – Client console



User is not authenticated!

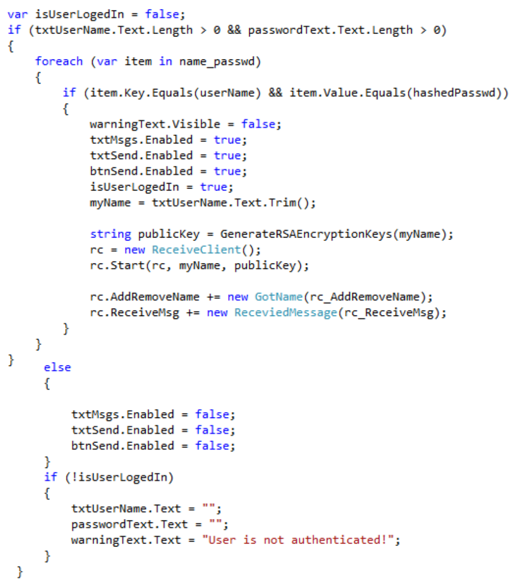
Fig. 5 UserName and Password part 3 – Error message

Password is encrypted with the hashing method, more specifically MD5CryptoServiceProvider() has been used. In the moment when user enters a name and password, and clicks LogIn, a method call is performed in which all registered users of the application are returned and cached in a Dictionary (name, password), then the hashing of the input password is performed and a check is done on the name, password existing pairs. If the password if found, then the program is continued, otherwise text field for name and password are reseted and an error message is shown.



```
private void btnLogin_Click(object sender, EventArgs e)
{
    //hash and save some passwords for test
    var userName = txtUserName.Text;
    var password = passwordText.Text;
    var hashedPasswd = hashPassword(password);
    Dictionary<string, string> name_passwd = new Dictionary<string, string>();
    string[] lines = System.IO.File.ReadAllLines(@"C:\\Users\\Danielusha\\
                                        Desktop\\ChatServer\\name_passwd.txt");
    foreach(var line in lines){
        var item = line.Split(',');
        name_passwd.Add(item[0],item[1]);
    }
```

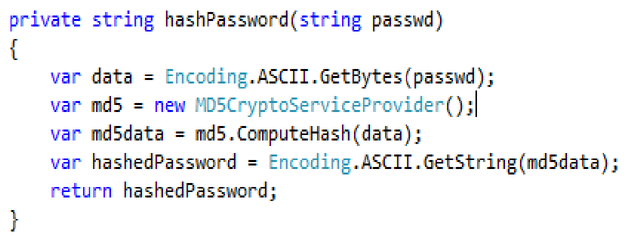
Fig. 6 UserName and Password part 4 – Login code



```
var isUserLogedIn = false;
if (txtUserName.Text.Length > 0 && passwordText.Text.Length > 0)
{
    foreach (var item in name_passwd)
    {
        if (item.Key.Equals(userName) && item.Value.Equals(hashedPasswd))
        {
            warningText.Visible = false;
            txtMsgs.Enabled = true;
            txtSend.Enabled = true;
            btnSend.Enabled = true;
            isUserLogedIn = true;
            myName = txtUserName.Text.Trim();

            string publicKey = GenerateRSAEncryptionKeys(myName);
            rc = new ReceiveClient();
            rc.Start(rc, myName, publicKey);

            rc.AddRemoveName += new GotName(rc_AddRemoveName);
            rc.ReceiveMsg += new ReceviedMessage(rc_ReceiveMsg);
        }
    }
}     else
      {

         txtMsgs.Enabled = false;
         txtSend.Enabled = false;
         btnSend.Enabled = false;
      }
      if (!isUserLogedIn)
      {
         txtUserName.Text = "";
         passwordText.Text = "";
         warningText.Text = "User is not authenticated!";
      }
 }
```

Fig. 7 Encryption and error message code

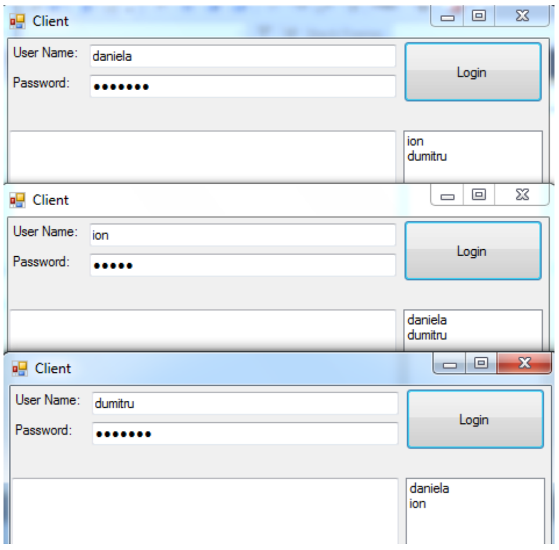
Hashing method (returns encrypted password as string):



```
private string hashPassword(string passwd)
{
    var data = Encoding.ASCII.GetBytes(passwd);
    var md5 = new MD5CryptoServiceProvider();
    var md5data = md5.ComputeHash(data);
    var hashedPassword = Encoding.ASCII.GetString(md5data);
    return hashedPassword;
}
```

Fig. 8 Hashing password code

If multiple users are connected, each of them will see the others in the graphical interface he was provided with, and can choose between all of them the one he wants to start a chat with.



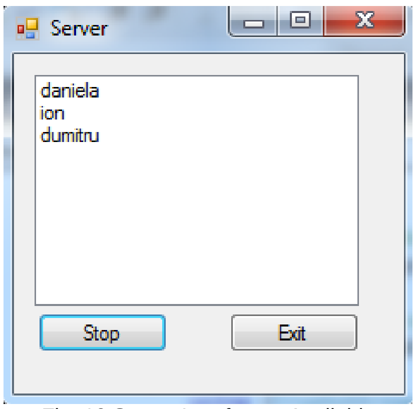
User Name: daniela
Password: •••••••
ion
dumitru

User Name: ion
Password: •••••
daniela
dumitru

User Name: dumitru
Password: •••••••
daniela
ion

Fig. 9 Client Interface – Available users

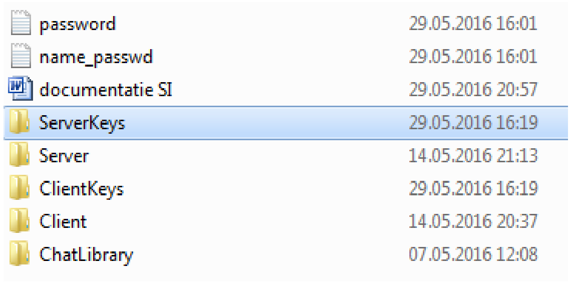
The same is for the server interface where all connected users will be visible.



daniela
ion
dumitru

Fig. 10 Server Interface – Available users

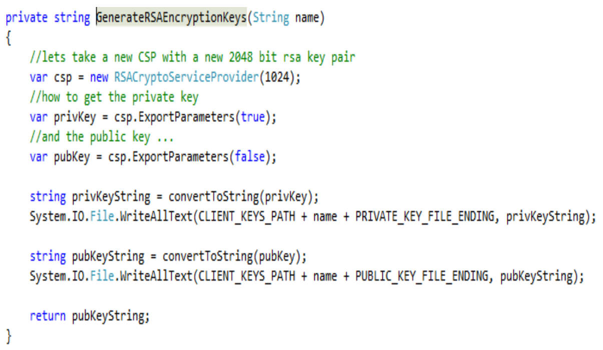
For messages encrypting there have been used two tipes of encrypting: asymethrical encrypting with a public key and RSA private key and symethrical AES encrypting. Both modalities have been used given RSA is the most secure encrypting method, but has the inconvenience that it can not be used on long messages. AES does the encrypting of the messages while RSA is used to encrypt the key and the VI of AES in order to send them both to the receiver of the message. Below methods defined for this behaviour have been described:
We create the publical key and the private key and we save them in xml files. For Server we have the folder ServerKeys, and each connected client will have a folder ClientKeys.



password	29.05.2016 16:01
name_passwd	29.05.2016 16:01
documentatie SI	29.05.2016 20:57
ServerKeys	29.05.2016 16:19
Server	14.05.2016 21:13
ClientKeys	29.05.2016 16:19
Client	14.05.2016 20:37
ChatLibrary	07.05.2016 12:08

Fig. 11 ServerKeys Folder

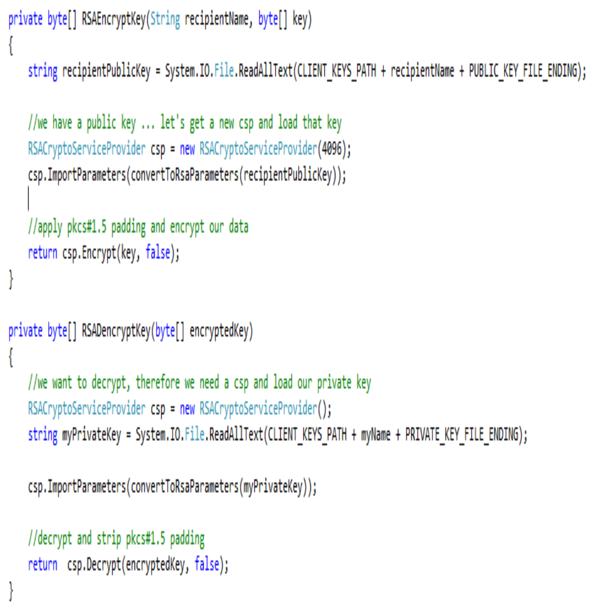
Generation of RSA key and caching in XML files is done by the code below.



```
private string GenerateRSAEncryptionKeys(String name)
{
    //lets take a new CSP with a new 2048 bit rsa key pair
    var csp = new RSACryptoServiceProvider(1024);
    //how to get the private key
    var privKey = csp.ExportParameters(true);
    //and the public key ...
    var pubKey = csp.ExportParameters(false);

    string privKeyString = convertToString(privKey);
    System.IO.File.WriteAllText(CLIENT_KEYS_PATH + name + PRIVATE_KEY_FILE_ENDING, privKeyString);

    string pubKeyString = convertToString(pubKey);
    System.IO.File.WriteAllText(CLIENT_KEYS_PATH + name + PUBLIC_KEY_FILE_ENDING, pubKeyString);

    return pubKeyString;
}
```

Fig. 12 Generation of RSA key and caching in XML files

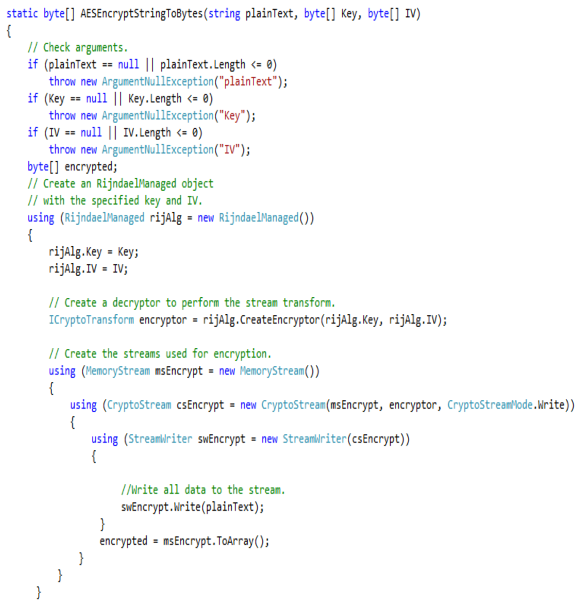
Encrypting and decrypting with RSA is done by the code below.



```
private byte[] RSAEncryptKey(String recipientName, byte[] key)
{
    string recipientPublicKey = System.IO.File.ReadAllText(CLIENT_KEYS_PATH + recipientName + PUBLIC_KEY_FILE_ENDING);

    //we have a public key ... let's get a new csp and load that key
    RSACryptoServiceProvider csp = new RSACryptoServiceProvider(4096);
    csp.ImportParameters(convertToRsaParameters(recipientPublicKey));

    //apply pkcs#1.5 padding and encrypt our data
    return csp.Encrypt(key, false);
}

private byte[] RSADencryptKey(byte[] encryptedKey)
{
    //we want to decrypt, therefore we need a csp and load our private key
    RSACryptoServiceProvider csp = new RSACryptoServiceProvider();
    string myPrivateKey = System.IO.File.ReadAllText(CLIENT_KEYS_PATH + myName + PRIVATE_KEY_FILE_ENDING);

    csp.ImportParameters(convertToRsaParameters(myPrivateKey));

    //decrypt and strip pkcs#1.5 padding
    return  csp.Decrypt(encryptedKey, false);
}
```

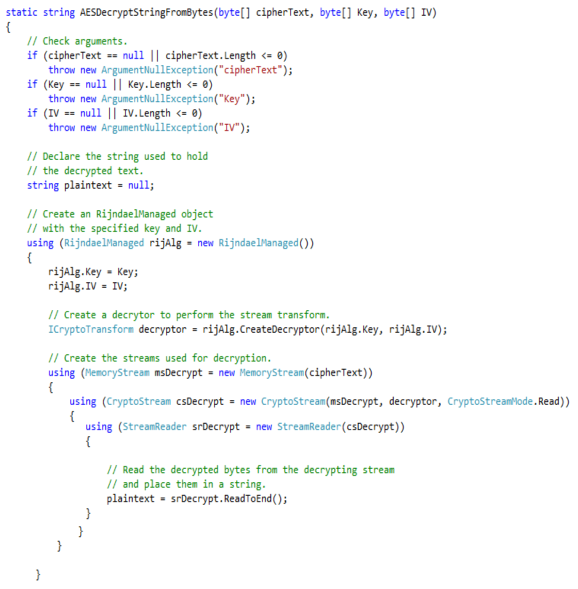
Fig. 13 Generation of RSA key and caching in XML files.

Encrypting and decrypting with AES is done by the code below.



```
static byte[] AESEncryptStringToBytes(string plainText, byte[] Key, byte[] IV)
{
    // Check arguments.
    if (plainText == null || plainText.Length <= 0)
        throw new ArgumentNullException("plainText");
    if (Key == null || Key.Length <= 0)
        throw new ArgumentNullException("Key");
    if (IV == null || IV.Length <= 0)
        throw new ArgumentNullException("IV");
    byte[] encrypted;
    // Create an RijndaelManaged object
    // with the specified key and IV.
    using (RijndaelManaged rijAlg = new RijndaelManaged())
    {
        rijAlg.Key = Key;
        rijAlg.IV = IV;

        // Create a decryptor to perform the stream transform.
        ICryptoTransform encryptor = rijAlg.CreateEncryptor(rijAlg.Key, rijAlg.IV);

        // Create the streams used for encryption.
        using (MemoryStream msEncrypt = new MemoryStream())
        {
            using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                {

                     //Write all data to the stream.
                     swEncrypt.Write(plainText);
                 }
                 encrypted = msEncrypt.ToArray();
             }
         }
     }
```

Fig. 14 Encrypting with AES

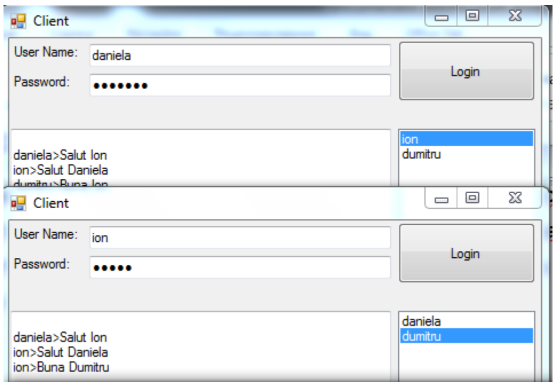


```
static string AESDecryptStringFromBytes(byte[] cipherText, byte[] Key, byte[] IV)
{
    // Check arguments.
    if (cipherText == null || cipherText.Length <= 0)
        throw new ArgumentNullException("cipherText");
    if (Key == null || Key.Length <= 0)
        throw new ArgumentNullException("Key");
    if (IV == null || IV.Length <= 0)
        throw new ArgumentNullException("IV");

    // Declare the string used to hold
    // the decrypted text.
    string plaintext = null;

    // Create an RijndaelManaged object
    // with the specified key and IV.
    using (RijndaelManaged rijAlg = new RijndaelManaged())
    {
        rijAlg.Key = Key;
        rijAlg.IV = IV;

        // Create a decrytor to perform the stream transform.
        ICryptoTransform decryptor = rijAlg.CreateDecryptor(rijAlg.Key, rijAlg.IV);

        // Create the streams used for decryption.
        using (MemoryStream msDecrypt = new MemoryStream(cipherText))
        {
            using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
            {
              using (StreamReader srDecrypt = new StreamReader(csDecrypt))
              {

                  // Read the decrypted bytes from the decrypting stream
                  // and place them in a string.
                  plaintext = srDecrypt.ReadToEnd();
              }
             }
          }

     }
```

Fig. 15 Decrypting with AES

Message transmission occurs in the following manner. User logs in, user creates its own public key and private RSA key and caches them locally, while the public key is forwarded to the server. Server writes the public key in an xml format file, caches it inside a directory and forwards all public keys to the other users connected at that time. Server will store all public keys of all users.

For being able to communicate each user will select from the list another user, will write the message. In the moment user will choose send option from the chat panel, the message will be encrypted using AES encrypting method, then by using the destinations public key RSA algorithm is applied to the AES message, the key and the VI. These data are now transmitted to the Server. Server receives the data, does not perform any action other than transferring them to the destination (name of the destination is also transmitted in order for the Server to be able to forward the message). The destination decrypts the data with the private key, then using the key and the VI it decrypts the whole AES encrypted message.



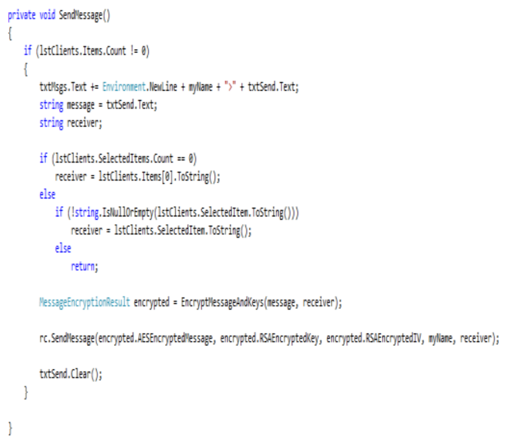
```
daniela
•••••••
daniela>Salut Ion
ion>Salut Daniela
dumitru>Buna Ion
```

```
ion
dumitru
```

```
ion
•••••
daniela>Salut Ion
ion>Salut Daniela
ion>Buna Dumitru
```

```
daniela
dumitru
```

Fig. 16 Conversations as shown to the users

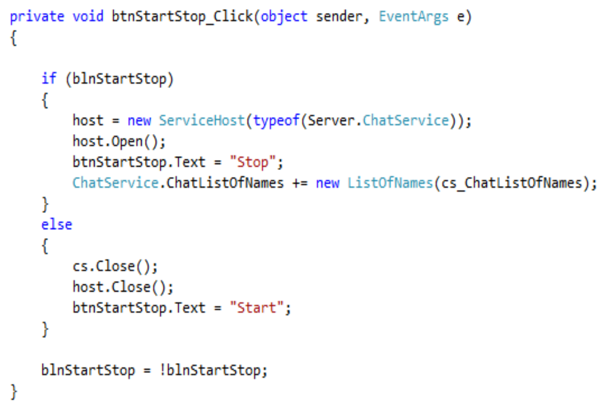


```
private void SendMessage()
{
    if (lstClients.Items.Count != 0)
    {
        txtMsgs.Text += Environment.NewLine + myName + ">" + txtSend.Text;
        string message = txtSend.Text;
        string receiver;

        if (lstClients.SelectedItems.Count == 0)
            receiver = lstClients.Items[0].ToString();
        else
            if (!string.IsNullOrEmpty(lstClients.SelectedItem.ToString()))
                receiver = lstClients.SelectedItem.ToString();
            else
                return;

        MessageEncryptionResult encrypted = EncryptMessageAndKeys(message, receiver);

        rc.SendMessage(encrypted.AESEncryptedMessage, encrypted.RSAEncryptedKey, encrypted.RSAEncryptedIV, myName, receiver);

        txtSend.Clear();
    }

}
```

Fig. 17 Send message code

The set of RSA keys are valid only during the program execution, once closed, the keys will no longer be valid. On a different connection new sets of keys will be created for the connected Client which will overwrite the existing ones inside the xml files.

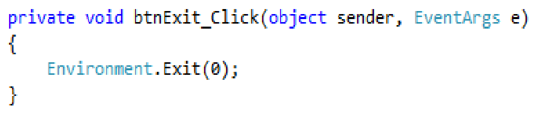
The server will be closed by selection of STOP button from the available interface having the following implementation:



```
private void btnStartStop_Click(object sender, EventArgs e)
{

    if (blnStartStop)
    {
        host = new ServiceHost(typeof(Server.ChatService));
        host.Open();
        btnStartStop.Text = "Stop";
        ChatService.ChatListOfNames += new ListOfNames(cs_ChatListOfNames);
    }
    else
    {
        cs.Close();
        host.Close();
        btnStartStop.Text = "Start";
    }

    blnStartStop = !blnStartStop;
}
```

Fig. 18 Stop Server code

which will stop the Server functioning, will delete the list of names from the graphical interface, and if user now will chose EXIT, Server will completely shut down.



```
private void btnExit_Click(object sender, EventArgs e)
{
    Environment.Exit(0);
}
```

Fig. 19 Exit code for server shutdown

III. CONCLUSIONS

In this paper there has been presented a research containg the development of an application client server on the chat.

In our own day it has a particular importance to study different types of implementations which can increase the security in communication.

A special interest can be observed in creating some possibilities to protect ourself during the transmition of messages. The whole project consists on two speciffic parts: a first one which developed the client server communication and the second one containing the encryption. While the first part consists much more in putting together some wellknown methods, the second part is the original part of this research. We combined modality through AES encryption messages and sent encrypted and the key message through a secure channel using RSA public key and private key. I also saved locally keys to all customers and we distributed among them by using the Server and save them in a local encrypted file. (File that contains only the password hash sites).

IV. ACKNOWLEDGEMENT

The infrastructure used for this work was partially supported by the project Integrated Center for research, development and innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for fabrication and control, Contract No. 671/09.04.2015, Sectoral Operational Program for Increase of the Economic Competitiveness co-funded from the European Regional Development Fund.

REFERENCES

[1] Rivest R., A. Shamir, L .Adleman, A method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21, pp.120-126, 1978.

[2] The Original RSA Patent as filed with the U.S. Patent Office by Rivest; Ronald L. (Belmont, MA), Shamir Adi (Cambridge, MA), Adleman; Leonard M. (Arlington, MA), December 14, 1977.

[3] Menezes A.J., van Oorschot P.C,Vanstone S.A.:Handbook of Applied Cryptography.The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N. W. Corporate Blvd., Boca Raton, FL 33431/9868, USA,1997.

[4] Gibson T., An architecture for Flexible Multi-Security Domain Networks, USA.

[5] Haque M. M., Pathan A.-S. K., Hong C. S., Securing U-Healthcare Sensor Networks using Public Key Based Scheme, *ICACT 2008*, February 17-20, 2008, pp. 1108-1111.

[6] Vimalathithan R., M. L. Valarmathi, Cryptanalysis of Simplified-DES using Computational Intelligence, TRANSACTIONS on COMPUTERS, Issue 6, Volume 10, July 2011, pp 210-219

[7] Daemen J., Rijmen V., "The Design of Rijndael: AES - The Advanced Encryption Standard", Springer-Verlag, 2002.

[8] M. A. Mioc, S. G. Pentiuc – Comparison between AES, Camellia and SEED, JMEST (Journal of Multidisciplinary Engineering Science and Technology) Vol.2 – Issue 12 (December – 2015).