P-1813

FRIEND TO FRIEND SYSTEM

A PROJECT REPORT

Submitted by

JASKIRAT SINGH VEEN	**71203104014**
MOHANA KUMARAN.S	**71203104019**
SRIDAR .R	**71203104051**

in partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY:: CHENNAI 600 025

APRIL 2007

P-1813

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"FRIEND TO FRIEND SYSTEM"** is the bonafide work of **"JASKIRAT SINGH VEEN, MOHANA KUMARAN.S, SRIDAR.R"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE** 21/4/07
Prof. Dr. S. Thangasamy	Ms.V. Vanitha.M.E.
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Computer Science and Engineering	Computer Science and Engineering
Kumaraguru College of Technology	Kumaraguru College of Technology
Coimbatore – 641 006	Coimbatore – 641 006

The candidates with University Register Numbers 71203104014, 71203104019, 71203104051 were examined by us in the project viva-voce examination held on 24/04/2007

INTERNAL EXAMINER

EXTERNAL EXAMINER 24/4/07

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Arutselvar Dr. N. Mahalingam, B.Sc, F.I.E.** and correspondent **Prof. K. Arumugam, B.E., M.S., M.I.E.,** for all their support and ray of strengthening hope extended.

We are immensely grateful to our principal **Dr. Joseph V. Thanikal, M.E., Ph.D., PDF., CEPIT.,** for his invaluable support to the come outs of this project.

We are extremely thankful to **Dr. S. Thangasamy.,** Head of the Department, Department of Computer Science and Engineering for his valuable advice and suggestions throughout this project,

We are indent to express our heartiest thanks to **Ms. S. Rajini, B.E.,** project coordinator and **Ms. V.Vanitha, M.E.,** guide who have helped us to making the project work extremely well.

We are also thankful to all the faculty members of the department of Computer Science and Engineering, Kumaraguru College of Technology, CBE for their valuable guidance, support and encouragement during the course of our project work.

We express our humble gratitude and thanks to our parents and family members who have supported and helped us to complete the project and our friends, for lending us valuable tips, support and co-operation throughout our project work.

ABSTRACT

The idea of restricting a network of computers to people you know, trust, or a closed group of users that require peer approval to allow entry of a new user into the network is the main theme of this application.

The concept of building a trust network and maintaining it can be done in many ways. We use a simple but effective technique of restricting access via referrals that are already a part of the network. Based on assumptions of trust that the invitation-only scheme imposes, the network may continue to evolve.

We aim to build an application with the following features that uses this type of a network of trust, and thus effectively facilitate features such as auto retrieval of files that any user of the network may send to you based on the fact that 'you trust the network'.

The application involves the following basic features for socializing or communication or data sharing.

- User Authentication to access the Application
- Registration of a new user based on a referral system
- Chatting in a public room with and interface similar to IRC

- Private messaging
- All communication that takes place is encrypted using Rijndael Algorithm using a session key basically just to avoid traffic analysis of any or all communication between users of the network/application.
- Transfer of data or media files between users
- DoodleBox , or more simply facilitation to share thumbnails and text scribbled among users into a box called the doodlebox and sent explicitly on user command
- Whiteboard facility to share a more real time Interaction between users based on mouse movements to a more pictorial interaction.

Apart from this the application is also aimed to be built with a user friendly interface that is efficient to use, Easier to learn and more satisfying to use.

TABLE OF CONTENTS

CH NO.	TITLE	PAGE NO.
	ABSTRACT	iv
	LIST OF FIGURES	viii
	LIST OF SYMBOLS	ix
1.	**INTRODUCTION**	1
	1.1. INVITATION ONLY NETWORKS	1
2.	**STREAMS AND PROTOCOLS**	3
3.	**THE INTERFACE**	4
	3.1. LOGIN/REGISTRATION INTERFACE	7
4.	**COMMUNICATION / MESSAGING**	8
	4.1. MANAGING THE RIJNDAEL ALGORITHM USING C#	9
5.	**THE DOODLEBOX**	10
	5.1. WORKING OF DOODLEBOX	11
	5.2. PROTOCOL TO SEND AND RECEIVE DOODLEBOX	12
6.	**FILE TRANSFER**	14
	6.1. PROTOCOL TO SEND AND RECEIVE FILES	16
	6.2. PLAYING MEDIA FILES IN WINDOWS MEDIA PLAYER	17
7.	**THE SERVER**	18
8.	**WHITEBOARDING**	19
	8.1. INTERFACE	20
	8.2. ARCHITECTURE	21
	8.3. MESSAGING ARCHITECTURE	23
	8.4. WORKING OF TOOLS	24
9.	**WORKING WITH THE WHITEBOARD**	26
10.	**APPENDIX 1- AES ALGORITHM**	28
	10.1. DEVELOPMENT	28
	10.2. DESCRIPTION OF THE CIPHER	28

CH NO.	TITLE	PAGE NO.
	10.2.1. THE ADDROUNDKEY STEP	30
	10.2.2. THE SUBBYTES STEP	30
	10.2.3. THE SHIFTROWS STEP	31
	10.2.4. THE MIXCOLUMNS STEP	32
	10.3. OPTIMIZATION OF THE CIPHER	33
11.	**APPENDIX 2 – THE SOCKETHELPER CLASS**	34
12.	**APPENDIX 3 – MANAGING RIJNDAEL ALGORITHM**	53
	REFERENCES	58

LIST OF FIGURES

3.1 GUI Interface – Client Window	5
3.2 Login/Registration Interface	7
6.1 Auto Retrieve feature	15
7.1 Server	18
8.1 Architecture -White boarding	21
8.2 Whiteboard Application Modes	22
8.3 Messaging Architecture	23
8.4 Pen Tool	24
8.5 Message Format	25
9.1 Server Mode	26
9.2 Client Mode	27

LIST OF SYMBOLS

SYMBOL/ ABREVIATION	MEANING	OCCURENCES (PAGE NOS)
FIFO	First In First Out	3
GUI	Graphical User Interface	5
IRC	Internet Relay Chat	1,4
AES	Advanced Encryption Standard	8,32-36
UI	User Interface	20
Jpg	Joint Photographic experts Group	20,46,49,50
IP	Internet Protocol	24,26
DES	Data Encryption Standard	32

1. INTRODUCTION

This application is designed to provide features like secure communication, file transfer, white boarding and DoodleBox or more simply a facility to share thumbnails and text scribbled among users into a box called the doodlebox and sent explicitly on user command all on an IRC like interface. The idea to make this system a network of trust is simply by making it an invitation only network.

1.1. INVITATION ONLY NETWORKS

An **invitation system** (or **invite system**) is one method of registration. Unlike open registration, where all users can join and closed registration, where there is a moratorium on new users joining, invites allow current users to select who can join, while allowing site administrators to maintain a stronghold on the population of the service.

Invitation systems are usually temporary. They are typically used for services in private beta testing, in order to control the number of users on the service. In other cases, they can be used due to limited availability of server resources. Rarely, they may be used on a permanent basis, in order to aggregate social network statistics (all users will ultimately have a traceable connection to all others). They can additionally be used to avoid trolling or spammers (all users trust all others), which is usually a positive side effect of the invitation system

Here we use invitation system for registration on a permanent basis so that a check on system is in place permanently. The concept is similar to referrals. Although we do not have a web based registration which would definitely be a future implementation, for the time being, registration is done during login time via the application itself. A new user is assumed to have got a copy from the peer who is inviting the new user and after the new user registers via the application interface to get new username and password, the referral would have to login and approve of the new user before the user can use the application in any way.

2. STREAMS AND PROTOCOLS

A stream is a continuous flow of bytes. For a chat server and a chat client to communicate over the network, each would read and write to the network stream. The network stream is duplex, and bytes transmitted and read are always in FIFO order.

When a chat client is connected to a chat server, they would have established a common network stream to use.

However, for any meaningful communication, there must be some rules and order. These rules and order would be known as the communication protocols. There are at least a few layers of protocols.

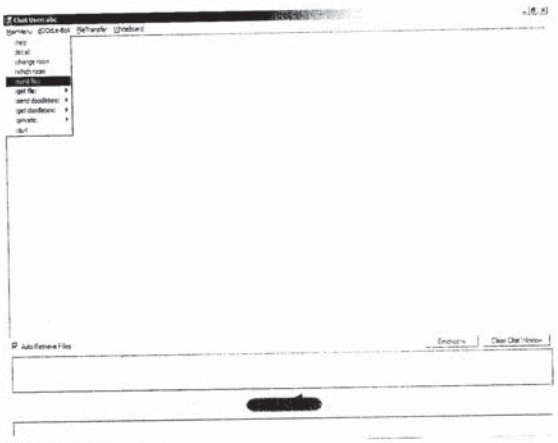
At the lowest level, the communicating parties would need to know how to break a continuous flow of bytes into packets/frames. These rules would be referred to as the Network Protocol.

At the next level, the parties would need to interpret the packets/frames. These rules would be known as the Application Protocol(s).

3. THE INTERFACE

The interface is similar to the IRC interface, which is more like a public chat room with number of users. **Internet Relay Chat (IRC)** is a form of real-time Internet chat or synchronous conferencing. It is mainly designed for group (many-to-many) communication in discussion forums called *channels*, but also allows one-to-one communication and data transfers via private message. IRC is based on a line-based structure with the client sending single-line messages to the server. receiving replies to those messages and receiving copies of some messages sent by other clients. In most clients, users can enter commands by prefixing them with '/', depending on the command these may either be passed directly to the server (generally used for commands the client does not recognize). passed to the server with some modification or handled entirely by the client.

The channels are not a part of this friend to friend system though. Commands are similar to the IRC, prefixed with ':', which can be typed in the client window to perform a particular action.



3.1 GUI Interface – Client Window

The various commands are

:help – *Displays a list of commands*

:change room- *Used to change from one public room to another*

:list – *Lists all users present in the current room*

:list all– *Lists all users present in the system anywhere*

:which room – *Tell the user which room he/she is in*

:private:<target>:<message> - *Used to send a private message to another user of the network*

:send doodlebox:<target> - *Used to send the doodlebox to another user*

:get doodlebox:<target> - *Used to get the doodlebox from another user who has proposed to send the same*

:quit – *Used to quit the application and the system*

3.1. LOGIN/REGISTRATION INTERFACE

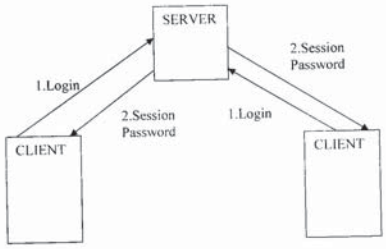
The following is the interface that is used for login/registration.

3.2 Login/Registration Interface

The users are stored in a file called *users.bin*, The file is nothing but a serialized hash table that contains the list of registered users of the system and is used to check upon during the login time. Registration is also done through the same interface; the new users can request the desired unique login name and password and should specify the referral who has invited the new user to the system. The new user will be allowed entry later only after the referral has approved of the new registration. To further limit the leeching growth of the network, a user can invite a new user only one at a time.

4. COMMUNICATION / MESSAGING

The Friend to friend system provides encrypted Communication, ensuring all shared information is known only to you and your trusted friends.
All messages are encrypted using the Rijndael[1] Algorithm. In cryptography, the **Advanced Encryption Standard (AES)**[1], also known as **Rijndael**, is a block cipher adopted as an encryption standard by the U.S. government. It has been analyzed extensively and is now used widely worldwide
The server allocates a session password for each room to all the clients every time a new client logs in or changes a room based on which the communication between the clients is encrypted.



Server sends session password at login time

1. Refer appendix 1 for a detailed description of the Rijndael / AES algorithm

4.1. MANAGING THE RIJNDAEL ALGORITHM USING C#

The C# language provides usage of various cryptic algorithms using the namespace Systems.Security.Cryptography.
The following snippet of code is used to implement a class that manages the Rijndael algorithm that will be later used to encrypt and decrypt messages when and where required
Refer appendix 3 for the source code to manage the Rijndael algorithm.

NOTE: Refer Appendix 1 for a detailed description of the Rijndael algorithm

5. THE DOODLEBOX

*A **doodle** is a mindless sketch, an aimless drawing, while a person's attention is otherwise occupied. Random thoughts are placed upon the paper during that time. Doodles can also just be sketches.*

The doodlebox in this application is an attempt to simulate a similar effect of doodling using the mouse, the doodlebox in this application also supports loading one or more pictures onto the doodlebox and simultaneously describe or doodle or communicate via pictorial representation which is done using the options in the Doodle Box menu or by simply right clicking on the doodlebox.

The doodlebox is sent to another user by usage of the following command in the client window.The console command to send the doodle box is:

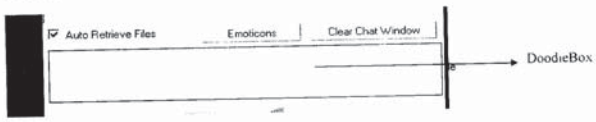
:send doodlebox: <target>

Where <target> refers to the username of a user who is currently online

Alternatively the user can make use of the Main menu to send the doodlebox to a particular user by choosing the receiver from the list in the send doodlebox sub menu.

5.1. WORKING OF DOODLEBOX

The Doodlebox used here is nothing but a picture box control in C#.



The concept is to write handlers for mouse move and mouse down event with respect to the picture box in the following manner to enable the doodling action in the doodle box.

Handler for Mouse move event of the doodlebox {

```
//If the left mouse button is used then return
    if(e.Button!=MouseButtons.Left)return;

//Else Create a new point which is the current point the mouse is on
    Point pt=new Point(e.X,e.Y);

//Draw a line from the previous point to the current point
    Graphics g=Graphics.FromImage(pic.Image);
    g.DrawLine(new Pen(Color.Black),pt1,pt);
//Mark this new point now as a start point which maybe used later
    pt1=pt;
//Refresh the Doodlebox
    pic.Refresh();
    }
```

Handler for mouse down event in the doodlebox {

```
//mark start point at mouse down for doodling further with mouse move
if(e.Button==MouseButtons.Left)
                    pt1=new Point(e.X,e.Y);

    }
```

5.2. PROTOCOL TO SEND AND RECEIVE DOODLEBOX

The protocol for sending a doodlebox is as follows:

- The client sends a command *:send doodlebox:<target>*.
- When the server receives the command, it will check if *<target>* has an active connection. It then replies with "*<server> send doodlebox*".
- When the client receives this special message, it will send a text message to indicate to the server the number of bytes of binary data that will be sent over. Following that, the binary data are then sent.
- The server uses the `ChatStream ReadBinary` method to get the binary data, and then saves the data to a file marked with the sender and target names.
- The server will then send a message to the *<target>* that there is a doodlebox ready for it to retrieve.

The protocol for getting a doodlebox is as follows:

- The client sends a command *:get doodlebox:<sender>*.
- When the server receives the command, it will first check if there is a file with the *<sender>* and the client name. If so, it will send the reply "*<server> get doodlebox*". It will then send a text message to indicate to the client the number of bytes to read. Then the binary data will be sent over.
- The client uses the common `ChatStream ReadBinary` method to get the binary data and display the image in the `RichTextBox`.

6. FILE TRANSFER

File transfer occurs on user command. It is actually a two step process where send file is a proposal to send a file and the get file command is the acknowledgement of the send file command. Any user of the system can send files to any other user who is available. The Auto retrieve feature is a useful add-on that enables auto retrieval of files without waiting for the receiver to manually respond to each send proposal.

The command to send a file is

:send file:<target>

Where <target> refers to the username of a user who is currently online

Alternatively user can make use of the menu to send a file to a particular user by choosing the receiver from the list under the send file menu.

The get file feature is used to get a file from a user who has proposed to send a file to the receiver.The command is

:get file:<from>

Where <from> refers to the username of a user who is currently online and has proposed to send a file

Alternatively user can make use of the menu to get a file from a particular user, who has proposed to send a file, by choosing the sender from the list under the send file menu.

The **Auto retrieve feature** if enabled does the work of getting the file automatically without waiting for the <receiver> to call for the file.
The 'get file' feature can be further used in cases when the receiver had not received a file due to some problem in transmission or some other error or simply because the receiver deleted the file after reception and now would like to get it again, At this point the use of 'get file' obtains the last file sent to the user by sender without disturbing the sender again to explicitly send the file.

6.1 Auto Retrieve feature

6.1. PROTOCOL TO SEND AND RECEIVE FILES

The protocol for transferring files is very similar to that for picture transfer. The main difference is that unlike a picture which is basically copied from a doodle box in the UI and saved to a file with a fixed *jpg* extension, the files are just tagged and stored by the program, and can have various different extensions. The extension for these files has to be maintained as the media player relies on the extension to play the files. We have used the client window to enable links to be recognized and clicked so that media files may be played using Windows media player.

When sending the binary data of a media clip to the server, the extension of the clip must be conveyed. And when the receiver retrieves the binary data from the server, the extension must also be made known so that the media clip file can be recreated with the correct extension.

To resolve this problem, there is a slight change in the protocol. When sending media clip data to the server, the sender first sends a three-character extension, followed by the number of bytes of binary data, and then finally, the binary data. The server first reads the extension, saves the extension to a file named *<sender> <receiver>* (without the extension), and then saves the binary data to a file named *<sender> <receiver>.<ext>*.

Similarly, when the receiver retrieves the media clip, the server first locates the file that stores the extension, retrieves the extension, and the retrieves the file *<sender>_<receiver>.<ext>*, and then sends the extension followed by the media clip binary data to the receiver.

6.2. PLAYING MEDIA FILES IN WINDOWS MEDIA PLAYER

If the file received is a media file and if the user wishes to play the file, all that the user has to do is to click on the link that is displayed on the client window.

To play the media clip, the system must have the media player installed. The client locates the media player from the Windows registry and sends the clip to be played by the media player.

7. THE SERVER

The server program starts on the port specified as the first argument, by creating the requested number of chat rooms specified by the second argument. It starts out the users by descrializing the *users.bin* file. Then, it continues to listen for connections. Once a connection is established, it sprouts a new `SocketHelper`[1] object to manage the communication with the connected client.

The SocketHelper[1] class contains methods to perform various actions on the server, namely the message transfer, file transfer, command interpretation and related operations. Nothing related with encryption apart from the password generation occurs on the server side. All the encryption and decryption work occurs on the client side.

7.1 Server

1. Refer Appendix 2 for a the SocketHelper Class source code

8. WHITEBOARDING

Whiteboards provide visual communication by allowing users to draw various shapes or lines for purposes of collaboration. Shapes can be manipulated and moved, and the session can even be saved as an image file. White boarding is an important feature in this Friend to Friend application to make interactive discussions within the peers through visual diagrams or drawing.

Any interactive discussion may require additional option apart from the usual textual exchanges. White boarding feature serves this purpose.

The idea is to capture mouse movements pixel by pixel and transmit them to the Whiteboard of its peers i.e. listeners as they occur live.

8.1. INTERFACE

The Whiteboard appears similar to the traditional paint application in Windows. The users work area is a white screen with toolbar on the top and connection panel at the bottom.

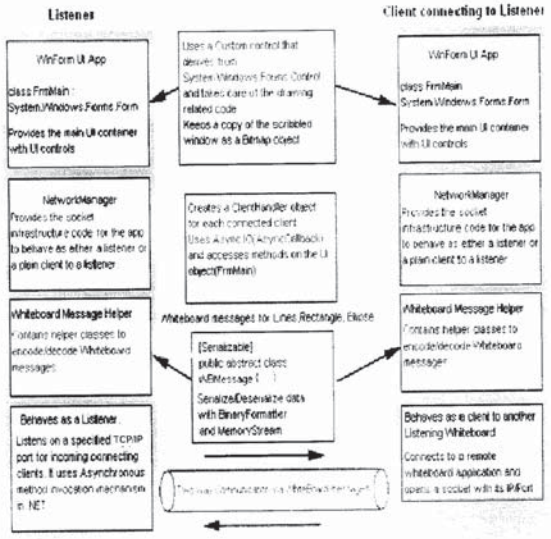
The toolbar contains the following tools

- Drawing Pen
- Rectangle auto shape
- Ellipse auto shape
- Eraser
- Save button

The connection panel contains the following:

- A tree structure showing the connected peers
- A radio button that enables the Whiteboard to behave either as a server or as a client
- Text boxes for specifying the port number and IP address

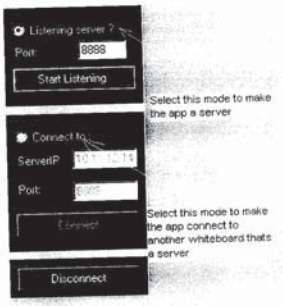
8.2. ARCHITECTURE



8.1 Architecture -White boarding

The Whiteboard has two modes of operation:

- Server mode (Listener)
- Client mode (Connects to a listener)



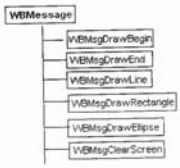
8.2 Whiteboard Application Modes

On selecting the first mode, as shown above, the application starts listening on the machine's IP on the port

On selecting the second option after specifying the correct IP/Port of another listening whiteboard, the application connects to the server.

8.3. MESSAGING ARCHITECTURE

Once the socket is established between the two users, each peer communicates with the other peer through Serialized Whiteboard Messages. This is an abstract class called `WBMessage` that every other message like `WBMsgDrawBegin`, `WBMsgDrawEnd`, `WBMsgDrawLine`, `WBMsgDrawRectangle` and `WBMsgDrawEllipse` inherits from.



8.3 Messaging Architecture

Serializable attribute is used before each of the classes. .NET provides object serialization support through the use of Formatter classes like the `BinaryFormatter` and `SoapFormatter`.

The application keeps transporting mouse messages (mousedown, mousemove and mouseup coordinates) from one user to another remote user. Using the `SoapFormatter` would have meant transporting a LOT more

data (since SOAP is an XML based serialization mechanism) than in a Binary Format.

`BinaryFormatters` serializes and deserializes an object or an entire graph of connected objects in binary format. For example, the following function serializes a long into a memory stream object whose raw buffer contents could be passed on the wire through the opened socket and then unpacked with a similar Deserialize routine.

8.4. WORKING OF TOOLS

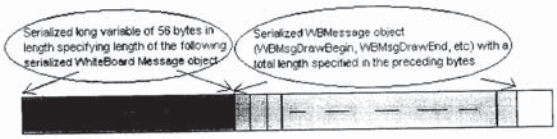
All the tools such as the drawing pen, ellipse tool and rectangle tool work in a similar fashion i.e. the only difference being the type of message and the shape appearing on the whiteboard. I.et us consider one such tool, the **drawing pen**, which works as follows

8.4 Pen Tool

The **DrawAreaCtrl** custom control shifts to the **WHITEBOARD_DRAW_MODE** of **enModeLine** Once this draw mode is enabled, all mouse events are interpreted for drawing related to scribbling lines.

MouseDown Event

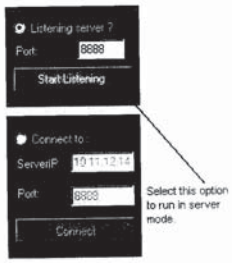
On a mousedown an instance of the serializable class WBMsgDrawBegin is created. This class is then serialized to a byte buffer using the BinaryFormatter class and sent across the socket. But every serialized object's buffer is preceded with a serialized long that specifies the length of the buffer that is going to follow it. This way the client on the other end knows how much data to parse and deserialize.



8.5 Message Format

9. WORKING WITH THE WHITEBOARD

- Run an instance of the application on one machine.
- Select the mode this instance would run in, i.e. client mode or server mode. To make it run as a server listener, select the "Start as Listener" option as shown and accept the default 8888 as the listening port, unless you have another application on your system listening on it, then click 'Start listening'.



9.1 Server Mode

- Run another instance of the Whiteboard application on another machine. This time select the 'Connect to' option as shown:

9.2 Client Mode

- The Connected Peers title on the top right would add a node with the peer's IP.

A user drawing on one end is reflected on the other, i.e. shown at the listener end. The whiteboard is locked when one user is drawing and can be used by the other user only when it becomes free. That is only one user at a time can annotate on the whiteboard.

10. APPENDIX 1- AES Algorithm

In cryptography, the **Advanced Encryption Standard** (**AES**), also known as **Rijndael**, is a block cipher adopted as an encryption standard by the U.S. government. It has been analyzed extensively and is now used widely worldwide as was the case with its predecessor, the Data Encryption Standard (DES). The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael", a combination of the names of the inventors.

10.1. DEVELOPMENT

Rijndael was a refinement of an earlier design by Daemen and Rijmen, Square; Square was a development from Shark.

Unlike its predecessor DES, Rijndael is a substitution-permutation network, not a Feistel network. AES is fast in both software and hardware, is relatively easy to implement, and requires little memory. As a new encryption standard, it is currently being deployed on a large scale.

10.2. DESCRIPTION OF THE CIPHER

Strictly speaking, AES is not precisely Rijndael (although in practice they are used interchangeably) as Rijndael supports a larger range of block and key sizes; AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits, whereas Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

The key is expanded using Rijndael's key schedule.

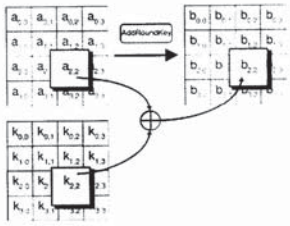
Most of AES calculations are done in a special finite field.

AES operates on a 4×4 array of bytes, termed the *state* (versions of Rijndael with a larger block size have additional columns in the state). For encryption, each round of AES (except the last round) consists of four stages:

1. AddRoundKey — each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.
2. SubBytes — a non-linear substitution step where each byte is replaced with another according to a lookup table.
3. ShiftRows — a transposition step where each row of the state is shifted cyclically a certain number of steps.
4. MixColumns — a mixing operation which operates on the columns of the state, combining the four bytes in each column using a linear transformation.

The final round replaces the MixColumns stage with another instance of AddRoundKey.

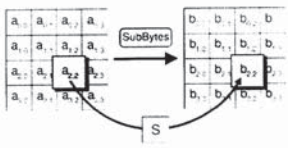
10.2.1. The AddRoundKey step



In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation (⊕).

In the AddRoundKey step, the subkey is combined with the state. For each round, a subkey is derived from the main key using Rijndael's key schedule; each subkey is the same size as the state. The subkey is added by combining each byte of the state with the corresponding byte of the subkey using bitwise XOR.

10.2.2. The SubBytes step

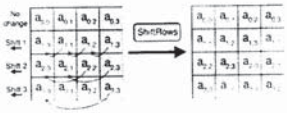


In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S; $b_{ij} = S(a_{ij})$.

In the SubBytes step, each byte in the array is updated using an 8-bit S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over $\mathbf{GF}(2^8)$, known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points (and so is a derangement), and also any opposite fixed points.

The S-box is more fully described in the article Rijndael S-box.

10.2.3. The ShiftRows step

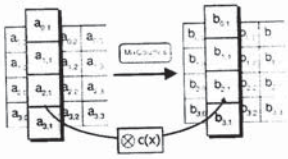


In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.

The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively. For the block of size 128 bits and 192 bits the shifting pattern is same. In this way, each column of the output state of the ShiftRows step is composed of

bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets). In the case of the 256 bit block, the first row is unchanged and the shifting for second, third and fourth row is 1 byte, 2 byte and 4 byte respectively - although this change only applies for the Rijndael cipher when used with a 256-bit block, which is not used for AES.

10.2.4. The MixColumns step



In the MixColumns step, each column of the state is multiplied with a fixed polynomial *c(x)*.

In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. The MixColumns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with ShiftRows, MixColumns provides diffusion in the cipher. Each column is treated as a polynomial over $\mathbf{GF}(2^8)$ and is then multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$. The MixColumns step can also be viewed as a matrix multiply in Rijndael's finite field.

This process is described further in the article Rijndael mix columns.

10.3. OPTIMIZATION OF THE CIPHER

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by converting the SubBytes, ShiftRows and MixColumns transformations into tables. One then has four 256-entry 32-bit tables, which utilizes a total of four kibibytes (4096 bytes) of memory--a kibibyte for each table. A round can now be done with 16 table lookups and 12 32-bit exclusive-or operations, followed by four 32-bit exclusive-or operations in the AddRoundKey step.

If the resulting four kibibyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit table by the use of circular rotates.

11. APPENDIX 2 – The SocketHelper Class

```
using System;
using System.Net.Sockets;
using System.Net;
using System.Threading;
using System.IO;
using System.Text;

namespace Chat
{
	//Delegate for actions
	public delegate void SocketHelperAction();

	public class SocketHelper
	{
		//reference chat server
		private ChatServer chatserver;
		//reference chat client
		private TcpClient client;
		//nickname
		private string nickname="";
		//room number 0 means no room assigned yet
		private int room=0;
		//data read from client
		private string readdata="";

		//action delegate
		private SocketHelperAction action;

		//public properties
		public int Room
		{
			get{return room;}
		}

		public string Nickname
		{
			get{return nickname;}
```



```
		}

		//2 parameters Constructor
		public SocketHelper(ChatServer s,TcpClient c)
		{
			client=c;
			chatserver=s;
			Thread t=new Thread(new ThreadStart(HandleClient));
			t.Start();
		}

		//Actions
		//DEFAULT
		private void action_default()
		{
			chatserver.Write(client.GetStream(),
ChatProtocolValues.UNKNOWN_CMD_MSG(readdata));
		}

		//MESSAGE
		private void action_send_message()
		{
			chatserver.Broadcast(nickname+"> " + readdata,room);
		}

		//PRIVATE MESSAGE
		private void action_private_message()
		{
			string[] s=readdata.Split(':');
			string name="null_name"; //give a default dummy name

			string temp=""; //hold the message
			//format is
			//:private:<target>:<message>
			if (s.Length>=3)name=s[2].Trim();
			if (s.Length>=4)temp=s[3].Trim();

			TcpClient t=null;
			if (chatserver.FindUserRoom(name)!=0)
```

```
t=(TcpClient)chatserver.ClientConnections[name.ToUpper()];

                if (t!=null)
                {
                        //to target
                        chatserver.Write(t.GetStream(),

ChatProtocolValues.NORMAL_MSG(nickname,temp));
                        //to inform sender
                        chatserver.Write(client.GetStream(),

        ChatProtocolValues.NORMAL_MSG(nickname,temp));
                }
                else
                        chatserver.Write(client.GetStream(),

ChatProtocolValues.USER_NOT_FOUND_MSG(name));
        }

        //LIST
        private void action_list()
        {
                string[] s=readdata.Split(' ');
                int p1=0; //default to unknown room
                if((s.Length)==1)p1=room; //set to current room

                //Get the specified room
                if(s.Length>=2)
                        if(s[1].ToUpper()=="ALL")
                                p1=-1; //LIST ALL: indicate all rooms
                        else
                                try
                                {
                                        p1=int.Parse(s[1]); //to get a room
number
                                }
                                catch{};

                if((p1>chatserver.NumRoom)||(p1==0))
```



```
                        chatserver.Write(client.GetStream(),

ChatProtocolValues.NO_SUCH_ROOM_MSG);
                else
                {
                        string temp="";



                        if(p1==-1)
                        {
                                for(int i=0;i<chatserver.NumRoom;i++)
                                        foreach(string s1 in
chatserver.RoomUsers[i].Values)
                                                temp=temp+"\n  "+s1 + " :
room " +(i+1);
                        }
                        else
                        {
                                foreach(string s1 in
chatserver.RoomUsers[p1-1].Values)
                                        temp=temp+"\n  "+s1 + " : room "
+(p1);
                        }

                        if (temp=="") temp="Empty";
                          chatserver.Write(client.GetStream(),

        ChatProtocolValues.NORMAL_MSG("server",temp));

                }
        }

        //WHICH ROOM
        private void action_which_room()
        {
                //Inform client of room number
                chatserver.Write(client.GetStream(),

        ChatProtocolValues.YOUR_ROOM_NO_MSG(room));
```



```
        }

        //QUIT
        private void action_quit()
        {
                //Inform client to quit the application
                chatserver.Write(client.GetStream(),
                        ChatProtocolValues.QUIT_MSG);
        }

        //CHANGE ROOM
        private void action_change_room()
        {
                //store old room number
                int oldroom=room;

                //Remove the user from the chat room

        chatserver.RemoveRoomUser(chatserver.RoomUsers[oldroom-
1],nickname);

                //Assigned to No room first
                room=0;

                //while no room is assigned
                while(room==0)
                {
                        //Get room number from client
                        chatserver.Write(client.GetStream(),

ChatProtocolValues.CHOOSE_ROOM(nickname,chatserver.NumRoom));
                        string temp=chatserver.Read(client.GetStream());

                        //check for valid room number
                        try
                        {
                                //convert the text message to integer
                                int r_n=int.Parse(temp);
                                //check to make sure that the room number
is within range
```



```
                                if ((r_n>=1) &&
(r_n<=chatserver.NumRoom))
                                        room=r_n;
                        }
                        catch{}
                }
                //Add user to the assigned room
                chatserver.AddRoomUser(chatserver.RoomUsers[room-
1],nickname);
                //Broadcast to old room participants

        chatserver.Broadcast(ChatProtocolValues.MOVE_TO(nickname,roo
m),oldroom);
                //Broadcast to new room participants

        chatserver.Broadcast(ChatProtocolValues.WELCOME(nickname,roo
m),room);

        }

        //HELP
        private void action_help()
        {
                chatserver.Write(client.GetStream(),
                        "server>\n"+
                        "  :help\n"+
                        "  :change room\n" +
                        "  :list\n"+
                        "  :list all\n"+
                        "  :list <room_no>\n"+
                        "  :which room\n"+
                        "  :private:<target>:<message>\n"+
                        "  :send doodlebox:<target>\n"+
                        "  :get doodlebox:<sender>\n" +
                        "  :quit" );
        }

        //SEND MEDIA
        private void action_send_media()
        {
```

```
                string[] s=readdata.Split(':');
                string name="";
                //format is
                //:send file:<target>
                if (s.Length==3)name=s[2];

                //Locate the target
                TcpClient t=null;
        if (chatserver.FindUserRoom(name)!=0)

        t=(TcpClient)chatserver.ClientConnections[name.ToUpper()];


                //If target is found
                if((t!=null))
                {
                        //Inform the sender(client) to send the media

        chatserver.Write(client.GetStream(),ChatProtocolValues.SEND_MED
IA_MSG);

                        string ext=chatserver.Read(client.GetStream());
                        //Find out the number of byte to read from sender
                        string
snumbytes=chatserver.Read(client.GetStream());
                int numbytes=int.Parse(snumbytes);

                if (numbytes==0){
                                chatserver.Write(client.GetStream(),
                                        "server> No media file to send");

                                return;
                }

                //must be less than 5 MB
                if (numbytes>5120000){
                                chatserver.Write(client.GetStream(),
                                        "server> Media File is larger than 5
MB");
```



```
                                return;
                }

                        //read the bytes
                        byte[]
b=chatserver.ReadBinary(client.GetStream(),numbytes);


                        if (b==null)
                        {
                                chatserver.Write(client.GetStream(),
                                        "server> Transmission Error");

                                return;
                        }

                        //To store the data in a file
                        //name convention is <sender>_<target>.ext

                        //create a file to hold the extension
                        FileStream fext=new FileStream(nickname
+"_"+name,FileMode.Create);
                        UnicodeEncoding Unicode = new
UnicodeEncoding();
                        byte[] bytes=Unicode.GetBytes(ext);
                fext.Write(bytes,0,bytes.Length );
                fext.Close();

                        FileStream f=new
FileStream(nickname+"_"+name+"."+ext,FileMode.Create);
                        f.Write(b,0,b.Length);
                        f.Close();
                        //Inform the target that there is a file from sender
                        chatserver.Write (t.GetStream(),

        ChatProtocolValues.MEDIA_FROM_MSG(nickname,name));
                        //Inform the sender that server had received the
media
                        chatserver.Write(client.GetStream(),
```



```
        ChatProtocolValues.MEDIA_SEND_MSG(nickname));
                }
                else
                {
                        //If target is not found inform sender
                        chatserver.Write(client.GetStream(),

        ChatProtocolValues.USER_NOT_FOUND_MSG(name));
                }
        }

        //GET MEDIA
        private void action_get_media()
        {
                string[] s=readdata.Split(':');
                string sender="";
                string medianame="";
                string ext="";
                //format is
                //:get file:<sender>:ext
                if(s.Length>=3)sender=s[2];
                if(s.Length>=4)ext=s[3];

                //format of saved jpg file is
                //<sender>_<target>.jpg
                //In this case the current user is the target

                //get the extension form the file

                if (File.Exists(sender + "_" + nickname))
                {
        FileStream f=new FileStream(sender + "_" +
nickname,FileMode.Open);
        FileInfo fi=new FileInfo(sender + "_" + nickname);
        byte[] b=new byte[fi.Length];
                        f.Read(b,0,b.Length);
        f.Close();
                        UnicodeEncoding Unicode = new
UnicodeEncoding();
```



```
                int charCount = Unicode.GetCharCount(b, 0,
b.Length);
                char[] chars = new Char[charCount];
                Unicode.GetChars(b, 0, b.Length, chars, 0);
                ext=new string(chars);

        }

        medianame=sender + "_" + nickname + "."+ext;

        //Check for existence of file
        if(!File.Exists(medianame))
                chatserver.Write(client.GetStream(),

        ChatProtocolValues.MEDIA_NOT_FOUND_MSG(medianame));
                else
                {
                        //Create a file stream
                        FileStream f=new
FileStream(medianame,FileMode.Open);
                        //To get the size of the file for purpose of memory
allocation
                        FileInfo fi=new FileInfo(medianame);
                        byte[] b=new byte[fi.Length];
                        //Read the content of the file and close
                        f.Read(b,0,b.Length);
                        f.Close();
                        //Inform the client to get the media
                        chatserver.Write (client.GetStream(),

        ChatProtocolValues.GET_MEDIA_MSG);
                        //Inform the client of the extension
                        //chatserver.Write(client.GetStream(),ext);

                        //Inform the client of the ext
                        chatserver.Write(client.GetStream(),ext);
                        //Inform the client of number of bytes
                        chatserver.Write(client.GetStream(),""+b.Length);
                        //Send the binary data
                        chatserver.WriteBinary(client.GetStream(),b);
```

```
                        //Inform the client that all binary data has been
send
                        chatserver.Write(client.GetStream(),

        ChatProtocolValues.MEDIA_SEND_ACK_MSG);

                        //Locate the sender of the media
                        TcpClient t=null;
                        if (chatserver.FindUserRoom(sender)!=0)

        t=(TcpClient)chatserver.ClientConnections[sender.ToUpper()];

                        //Inform the sender that the target has gotten the
media
                        if(t!=null)
                                chatserver.Write(t.GetStream(),

        ChatProtocolValues.GOTTEN_MEDIA_MSG(nickname));

                }
        }


        //SEND PIC
        private void action_send_pic()
        {
                string[] s=readdata.Split(':');
                string name="";
                //format is
                //:send doodlebox:<target>
                if (s.Length==3)name=s[2];

                //Locate the target
                TcpClient t=null;
    if (chatserver.FindUserRoom(name)!=0)

    t=(TcpClient)chatserver.ClientConnections[name.ToUpper()];


                //If target is found
```



```
                if((t!=null))
                {
                        //Inform the sender(client) to send the picture

        chatserver.Write(client.GetStream(),ChatProtocolValues.SEND_PIC_
MSG);

                        //Find out the number of byte to read from sender
                        string
snumbytes=chatserver.Read(client.GetStream());
                int numbytes=int.Parse(snumbytes);

                        //read the bytes
                        byte[]
b=chatserver.ReadBinary(client.GetStream(),numbytes);
                        if (b==null)
                        {
                                chatserver.Write(client.GetStream(),
                                        "server> Transmission Error");

                                return;
                        }

                        //To store the data in a jpg file
                        //name convention is <sender>_<target>.jpg
                        FileStream f=new
FileStream(nickname+"_"+name+".jpg",FileMode.Create);
                        f.Write(b,0,b.Length);
                        f.Close();
                        //Inform the target that there is a picture from
sender
                        chatserver.Write (t.GetStream(),

                ChatProtocolValues.PIC_FROM_MSG(nickname,name));
                        //Inform the sender that server had received the
picture
                        chatserver.Write(client.GetStream(),

                ChatProtocolValues.PIC_SEND_MSG(nickname));
                        }
```



```
                else
                {
                        //If target is not found inform sender
                        chatserver.Write(client.GetStream(),

        ChatProtocolValues.USER_NOT_FOUND_MSG(name));
                }
        }

        //GET PIC
        private void action_get_pic()
        {
                string[] s=readdata.Split(':');
                string sender="";
                string picname="";
                //format is
                //:get doodlebox:<sender>
                if(s.Length==3)sender=s[2];

                //format of saved jpg file is
                //<sender>_<target>.jpg
                //In this case the current user is the target
                picname=sender + "_" + nickname + ".jpg";

                //Check for existence of file
                if(!File.Exists(picname))
                        chatserver.Write(client.GetStream(),

        ChatProtocolValues.PIC_NOT_FOUND_MSG(picname));
                else
                {
                        //Create a file stream
                        FileStream f=new
FileStream(picname,FileMode.Open);
                        //To get the size of the file for purpose of memory
allocation
                        FileInfo fi=new FileInfo(picname);
                        byte[] b=new byte[fi.Length];
                        //Read the content of the file and close
                        f.Read(b,0,b.Length);
```



```
                        f.Close();
                        //Inform the client to get the pic
                        chatserver.Write (client.GetStream(),

        ChatProtocolValues.GET_PIC_MSG);
                        //Inform the client of number of bytes
                        chatserver.Write(client.GetStream(),""+b.Length);
                        //Send the binary data
                        chatserver.WriteBinary(client.GetStream(),b);
                        //Inform the client that all binary data has been
send
                        chatserver.Write(client.GetStream(),

        ChatProtocolValues.PIC_SEND_ACK_MSG);

                        //Locate the sender of the picture
                        TcpClient t=null;
                        if (chatserver.FindUserRoom(sender)!=0)

        t=(TcpClient)chatserver.ClientConnections[sender.ToUpper()];

                        //Inform the sender that the target has gotten the
picture
                        if(t!=null)
                                chatserver.Write(t.GetStream(),

        ChatProtocolValues.GOTTEN_PIC_MSG(nickname));

                }
        }


        //The main method that handle all the chat communication with
the client
        private void HandleClient()
        {
                try
                {
                        //Autheticate the user
```

```
                    AuthenticationServer auth=new
AuthenticationServer(chatserver.client);
                    if(!auth.Authenticate())
                        throw(new Exception());

                    //Send connection message to client
                    //returning the user id
                    chatserver.Write(client.GetStream(),

    ChatProtocolValues.CONNECTION_MSG(auth.UserID));

                    //buffer
                    byte[] bytes= new byte[256];

                    //set nickname as user id which is alraedy unique
                    nickname=auth.UserID;

                    //Assign a room to connected client
                    room=chatserver.AssignRoom(nickname);
                    Console.WriteLine("room assigned= "+room + "
for " + nickname);

                    //Add to Connections
                    try
                    {
                        chatserver.AddConnection(nickname,client);
                    }
                    catch{}

                    //Broadcast to chat room about the new user

    chatserver.Broadcast(ChatProtocolValues.WELCOME(nickname,roo
m),room);

                    //listen to all client data send

    while((readdata=chatserver.Read(client.GetStream()))!="")
                    {
                        readdata=readdata.Trim();
```



```
                    //Check if the readdata is a command

    if(readdata.ToUpper().Substring(0,1)==ChatProtocolValues.IS_CMD
)
                    {
                        //Assign a default action
                        action=new
SocketHelperAction(action_default);

                        //Reassign action based on format
content
                        if
((readdata.ToUpper()+":").IndexOf(ChatProtocolValues.GET_PIC_CMD)=
=0)
                            action=new
SocketHelperAction(action_get_pic);

                        if
((readdata.ToUpper()+":").IndexOf(ChatProtocolValues.SEND_PIC_CMD)
==0)
                            action=new
SocketHelperAction(action_send_pic);

                        if
((readdata.ToUpper()+":").IndexOf(ChatProtocolValues.GET_MEDIA_CM
D)==0)
                            action=new
SocketHelperAction(action_get_media);

                        if
((readdata.ToUpper()+":").IndexOf(ChatProtocolValues.SEND_MEDIA_C
MD)==0)
                            action=new
SocketHelperAction(action_send_media);


                        if ((readdata.ToUpper()+"
").IndexOf(ChatProtocolValues.HELP_CMD)==0)
```



```
                            action=new
SocketHelperAction(action_help);

                        if ((readdata.ToUpper()+"
").IndexOf(ChatProtocolValues.QUIT_CMD)==0)
                            action=new
SocketHelperAction(action_quit);

                        if ((readdata.ToUpper()+"
").IndexOf(ChatProtocolValues.CHANGE_ROOM_CMD)==0)
                            action=new
SocketHelperAction(action_change_room);

                        if ((readdata.ToUpper()+"
").IndexOf(ChatProtocolValues.WHICH_ROOM_CMD)==0)
                            action=new
SocketHelperAction(action_which_room);

                        if ((readdata.ToUpper()+"
").IndexOf(ChatProtocolValues.LIST_CMD)==0)
                            action=new
SocketHelperAction(action_list);

                        if
((readdata.ToUpper()+":").IndexOf(ChatProtocolValues.PRIVATE_MSG_C
MD )==0)
                            action=new
SocketHelperAction(action_private_message);
                    }
                    else //NON COMMAND
                    {
                //Print out read data in console
                Console.WriteLine("Read>" + readdata);
                        //if not a command assign to a mesage
sending action
                        action=new
SocketHelperAction(action_send_message);
                    } //COMMANDS

                    //perform the action
```



```
                    action();
                }//WHILE

            }
            catch(Exception e)
            {
                //Trapped exception
                Console.WriteLine("The following error is trapped
by the chat server");
                Console.WriteLine(e);

                Console.WriteLine("Waiting for Connection...");

            }
            finally
            {
                //while loop ended or when there are some other
problems
                //try to inform client to shut down
                try
                {

    chatserver.Write(client.GetStream(),ChatProtocolValues.QUIT_MSG
);
                }
                catch{}

                //if the client had belong to a room
                if ((room!=0) && (nickname!=""))
                {
                    //remove user from room

    chatserver.RemoveRoomUser(chatserver.RoomUsers[room-
1],nickname);
                    //inform all that the client has logged out

    chatserver.Broadcast(ChatProtocolValues.USER_LOG_OUT(nickna
me,room));
                }
```

```
                    //remove the client connection
                    try
                    {
    chatserver.RemoveConnection(nickname,client);
                    }
                    catch{}

                }
            }

        }
}
```


12. APPENDIX 3 – Managing Rijndael Algorithm

//Source code to manage Rijndael Algorithm using C#

```
public sealed class Cryption
{

  private RijndaelManaged Algorithm;
  //memory stream
  private MemoryStream memStream;
  //ICryptoTransform interface
  private ICryptoTransform EncryptorDecryptor;
  //CryptoStream
  private CryptoStream crStream;
  //Stream writer and Stream reader
  private StreamWriter strWriter;
  private StreamReader strReader;
  //internal members
  private string m_key;
  private string m_iv;
  //the Key and the Inicialization Vector
  private byte[] key;

  private byte[] iv;
  //password view
  private string pwd_str;
  private byte[] pwd_byte;
  //Constructor
```



```
public Cryption(string key_val, string iv_val)
{
  key = new byte[32];
  iv = new byte[32];

  int i;
  m_key = key_val;
  m_iv = iv_val;
  //key calculation, depends on first constructor parameter
  for (i = 0; i < m_key.Length; i++)
  {
    key[i] = Convert.ToByte(m_key[i]);
  }
  //IV calculation, depends on second constructor parameter
  for (i = 0; i < m_iv.Length; i++)
  {
    iv[i] = Convert.ToByte(m_iv[i]);
  }

}
//Encrypt method implementation
public string Encrypt(string s)
{
  //new instance of algorithm creation
  Algorithm = new RijndaelManaged();

  //setting algorithm bit size
```



```
  Algorithm.BlockSize = 256;
  Algorithm.KeySize = 256;

  //creating new instance of Memory stream
  memStream = new MemoryStream();

  //creating new instance of the Encryptor
  EncryptorDecryptor = Algorithm.CreateEncryptor(key, iv);

  //creating new instance of CryptoStream
  crStream = new CryptoStream(memStream, EncryptorDecryptor,
  CryptoStreamMode.Write);

  //creating new instance of Stream Writer
  strWriter = new StreamWriter(crStream);

  //cipher input string
  strWriter.Write(s);

  //clearing buffer for currnet writer and writing any
  //buffered data to //the underlying device
  strWriter.Flush();
  crStream.FlushFinalBlock();

  //storing cipher string as byte array
  pwd_byte = new byte[memStream.Length];
  memStream.Position = 0;
```

```
    memStream.Read(pwd_byte, 0, (int)pwd_byte.Length);

    return Convert.ToBase64String(pwd_byte);
  }

  //Decrypt method implementation
  public string Decrypt(string s)
  {
    //new instance of algorithm creation
    Algorithm = new RijndaelManaged();

    //setting algorithm bit size
    Algorithm.BlockSize = 256;
    Algorithm.KeySize = 256;

    //creating new Memory stream as stream for input string
    MemoryStream memStream = new MemoryStream(
      Convert.FromBase64String(s));

    //Decryptor creating
    ICryptoTransform EncryptorDecryptor =
      Algorithm.CreateDecryptor(key, iv);

    //setting memory stream position
    memStream.Position = 0;

    //creating new instance of Crupto stream
```



```
    CryptoStream crStream = new CryptoStream(
      memStream, EncryptorDecryptor, CryptoStreamMode.Read);

    //reading stream
    strReader = new StreamReader(crStream);

    //returnig decrypted string
    return strReader.ReadToEnd();
  }
}
```


REFERENCES

1. Bruce Schneier (1996) – Applied Cryptography
2. Karli Watson , Christian Nagel , Eric White , Jacob Hammer Pedersen (Author), Jon Reid , Matthew Reynolds , Morgan Skinner , Zach Greenvoss , David Espinosa (2005) – Beginning Visual C# 2005
3. Advanced Encryption Standard, http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
4. http://msdn2.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged.aspx
5. MSDN Online Documentation