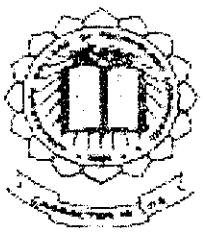
P-3301



EFFICIENT SECURE MESSAGE ROUTING FOR STRUCTURED PEER TO PEER SYSTEMS

A PROJECT REPORT

Submitted by

ABISHEK PAUL JACKSON.F	71206205002
SRIHARIPRASATH.B	71206205047
SRINIVASAN.M	71206205048

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641006

ANNA UNIVERSITY: CHENNAI 600 025

April 2010

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Efficient secure message routing for structured peer to peer systems**" is the bonafide work of

ABISHEK PAUL JACKSON.F	**71206205002**
SRIHARIPRASATH.B	**71206205047**
SRINIVASAN.M	**71206205048**

who carried out the project work under my supervision.

SIGNATURE

Ms.N.Suganthi, M.E., Ph.D.
Assistant Professor,
Department of Information Technology,
Kumaraguru College of Technology,
Chinnavedampatti Post,
Coimbatore-641006.

SIGNATURE

Dr.L.S.Jayashree, Ph.D..,
Head of Department,
Department of Information Technology,
Kumaraguru College of Technology,
Chinnavedampatti Post,
Coimbatore-641006.

Submitted for viva-voice examination held on 19/04/10.

Internal Examiner 19/4/10

External Examiner 19.4.10

DECLARATION

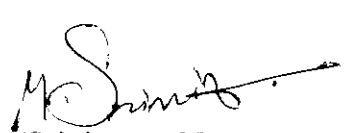
We hereby declare that the project entitled " **EFFICIENT SECURE MESSAGE ROUTING FOR STRUCTURED PEER TO PEER SYSTEMS**" is a record of original work done by us and to the best our knowledge, a similar work has not been submitted to Anna University or any institutions, for fulfillment of course study.

The report is in partial fulfillment of the requirements for the award of the degree of Bachelor of Information Technology of Anna University, Chennai.

Place: Coimbatore
Date: 16-04-2010

(Abishek Paul Jackson.F)

(Sriharipraśath.B)



(Srinivasan.M)

ACKNOWLEDGEMENT

The exhilaration achieved on successful completion of any task should be shared with the people behind the venture. At the onset, we thank the management of our college for having provided the excellent facilities to carry out the project.

We express our deep gratitude to our principal Dr.S.Ramachandran for ushering us in the path of triumph.

We are always thankful to our beloved Professor, Dean and HEAD of Computer Science and Engineering Department, Dr.S.Thangasamy, whose consistent support and enthusiastic involvement helped us a great deal.

We convey our sincere thanks to our project coordinator Professor Ms.L.S.Jayashree, for her invaluable assistance.

We are greatly indebted to our beloved guide Ms.N.Suganthi, assistant professor, Department of Computer Science and Engineering for her excellent guidance and timely support during the course of this project.

We also feel elated in manifesting our deep sense of gratitude to all the staff and lab technicians in the Department of Information Technology

ABSTRACT

Structured peer-to-peer overlay networks provide a substrate for the construction of large-scale, decentralized applications, including distributed storage, group communication, and content distribution. These overlays are highly resilient; they can route messages correctly even when a large fraction of the nodes crash or the network partitions. But current overlays are not secure; even a small fraction of malicious nodes can prevent correct message delivery throughout the overlay. This problem is particularly serious in open peer-to-peer systems, where many diverse, autonomous parties without preexisting trust relationships wish to pool their resources.

To studies attacks aimed at preventing correct message delivery in structured peer-to-peer overlays and presents defenses to these attacks. We describe and evaluate techniques that allow nodes to join the overlay, to maintain routing state, and to forward messages securely in the presence of malicious nodes. We consider security issues in structured p2p overlay networks. We describe attacks that can be mounted against such overlays and the applications they support, and present the design of secure techniques that can thwart such attacks. In particular, we identify secure *routing* as a key building block that can be combined with existing, application-specific security techniques to construct secure, decentralized applications upon structured overlays. Secure routing requires (1) a secure assignment of node identifiers, (2) secure routing table Maintenance and (3) secure message forwarding.

We present techniques for each of these problems, and show how using these techniques, secure routing can be maintained efficiently despite up to 25% of malicious participating nodes. Moreover, we show that the overhead of secure routing is acceptable and proportional to the fraction of malicious nodes.

TABLE OF CONTENTS

LIST OF FIGURES

1.INTRODUCTION

1 INTRODUCTION

1.1 INTRODUCTION

In structured P2P systems, peers and keys of content objects are identified using a set of well defined IDs. Queries are routed towards the target with a certain ID. When one or more nodes are malicious, they may prevent correct message routing. Alternate routing paths can be used to circumvent them. As a result, the routing latency consists of two parts: normal routing latency and extra routing latency incurred by bypassing malicious nodes. In this paper, we propose tracer routing, an efficient routing strategy designed to control the routing path while reducing the normal routing latency. Combined with a peer-ID based signature scheme, it can offer the initiator of each query to identify malicious nodes. A key feature of our scheme from other protocols is that alternate routing is constructed only detecting malicious nodes. Our simulation shows that the routing success rate our scheme can achieve is better than previous protocols.

We define a secure routing primitive that can be combined with existing techniques to construct secure applications on structured p2p overlays. Subsequent sections show how to implement the secure routing primitive under the fault and network models that we described in the previous section. The routing primitives implemented by current structured p2p overlays provide a best-effort service to deliver a message to a replica root associated with a given key. With malicious overlay nodes, the message may be dropped or corrupted, or it may be delivered to a malicious node instead of a legitimate replica root. Therefore, these primitives cannot be used to construct secure applications.

For example, when inserting an object, an application cannot ensure that the replicas are placed on legitimate, diverse replica roots as opposed to faulty nodes that impersonate replica roots. Even if applications use cryptographic methods to authenticate objects, malicious nodes may still corrupt, delete, deny access to or supply stale copies of all replicas of an object Secure routing table maintenance ensures that the fraction of faulty nodes that appear in the routing tables of correct nodes does not exceed, on average, the fraction of faulty nodes in the entire overlay. Without it, an attacker could prevent correct message delivery, given only a relatively small number of faulty nodes. Finally, secure message forwarding ensures that at least one copy of a message sent to a key reaches each correct replica root for the key with high probability.

1.2 **RELATED WORK**

Previous works on secure message routing concentrated mainly on employing redundancy to increase the probability of a message being delivered successfully the target. The goal of message routing is to maximize the probability of at least one copy of the correct message reach the correct target.

Initiator sends a query to all of its neighbors in Pastry overlay. Then each neighbor forwards the query toward the target. If at least one copy of query arrive target, the query is considered successfully delivered. However, it will Inevitably cause congestion and burden system, especially in a bandwidth intensive system. Let c be the bandwidth cost of sending a query from a peer to one of its neighbors, l be the number of neighbors of each peer and h be the average number of hops of each query. Without any message redundancy, the average bandwidth cost of each query is only $c \times h = O(h)$. But with message redundancy, the average bandwidth cost is increased to $c \times (l + l2 + ... + lh)$

2.SYSTEM ANALYSIS

$O(lh)$. It provides d independent paths between two peers in the P2P system. However, the routing success rate degrades greatly as n is a big value, where n denotes the number of nodes in the P2P system.

2 SYSTEM ANALYSIS

2.1 ROUTING IN P2P SYSTEMS:

The routing primitives implemented by current structured p2p overlays provide a best effort service to deliver a message to a replica root associated with a given key. As discussed above, a malicious overlay node has ample opportunities to corrupt overlay level communication. Therefore, these primitives are not sufficient to construct secure applications. For example, when inserting an object, an application cannot ensure that the replicas are placed on legitimate, diverse replica roots as opposed to faulty nodes that impersonate replica roots. Even if applications use cryptographic methods to authenticate objects, malicious nodes may still corrupt, delete, deny access to or supply stale copies of all replicas of an object.

To address this problem, we must create a secure routing primitive. The secure routing primitive ensures that when a non-faulty node sends a message to a key k, the message reaches all non-faulty members in the set of replica roots Rk with very high probability. Rk is defined as the set of nodes that contains, for each member of the set of replica keys associated with k, a live root node that is responsible for that replica key. In Pastry, for instance, Rk is simply a set of live nodes with nodeIds numerically closest to the key. Secure routing ensures that the message is eventually delivered, despite nodes that may corrupt, drop or misroute

the message; and the message is delivered to all legitimate replica roots for the key, despite nodes that may attempt to impersonate a replica root.

Secure routing can be combined with existing security techniques to safely maintain state in a structured p2p overlay. For instance, self-certifying data can be stored on the replica roots, or a Byzantine-fault-tolerant replication algorithm can be used to maintain the replicated state. Secure routing guarantees that the replicas are initially placed on legitimate replica roots, and that a lookup message reaches a replica if one exists. Similarly, secure routing can be used to build other secure services, such as maintaining file metadata and user quotas in a distributed storage utility.

Implementing the secure routing primitive requires the solution of three problems:

securely assigning nodeIds to nodes, securely maintaining the routing tables, and securely forwarding messages. Secure nodeId assignment ensures that an attacker cannot choose the value of nodeIds assigned to the nodes that the attacker controls. Without it, the attacker could arrange to control all replicas of a given object, or to mediate all traffic to and from a victim node. Secure routing table maintenance ensures that the fraction of faulty nodes that appear in the routing tables of correct nodes does not exceed, on average, the fraction of faulty nodes in the entire overlay. Without it, an attacker could prevent correct message delivery, given only a relatively small number of faulty nodes. Finally, secure message forwarding ensures that at least one copy of a message sent to a key reaches each correct replica root for the key with high probability.

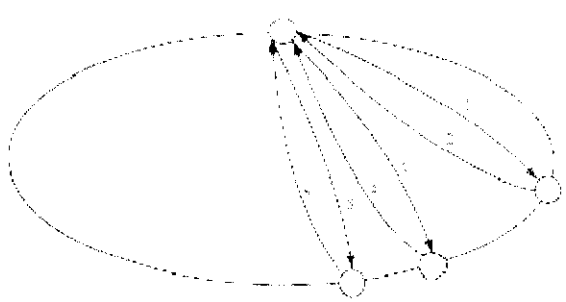
2.2 SEMATIC APPROACHES

2.2.1 Iterative routing:

In iterative routing, the initiator can know the whole routing process. In each step, the initiator asks intermediate peer x to return the IP address of the next hop, instead of letting x forward the query directly. With the returned IP address, the initiator sends the query to each peer of the routing path on its own, repeating until the query reaches the target.

Nodes communicate only with the originator. The tick mark denotes the position of the queried key in the ID space. The square shows the key's successor node. The Triangles and arrows show a lookup path. The last node before the tick mark is the key's predecessor.

Iterative routing is not efficient, but it gains some benefits due to its manageable behavior. The intermediate peers reply with the IP address of the next hops to the initiator, the initiator can send the message to the peer in the next hop directly.

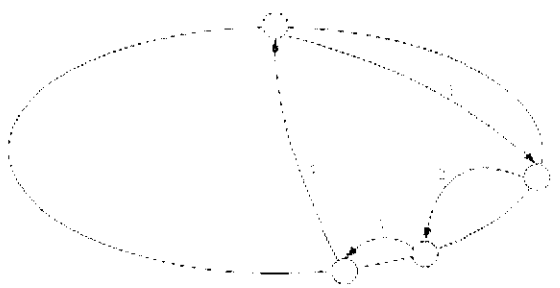


Iterative Routing

O – Node 1,3,5-Encrypted Message & Next host IP address 2,4,6 - ACK

2.2.2 Recursive routing:

With recursive routing, the initiator issues a query to the nearest peer to the target according to its routing table. If the intermediate peer, say x, is not the target, x will forward the query to the peer in the next hop, without making any acknowledgement to the initiator. The process repeats until query reaches the target. The target sends the query result back to the initiator directly. Recursive routing enable routing queries as quickly as possible, but the initiator has no control over the routing process.



Recursive Routing

O - Node 1,2,3-Encrypted Message & Destination IP address 4- ACK

We consider three kinds of attacks:

In case 1, the intermediate peer x pollutes or forges the content of the query. The next hop will still receive the original query since initiator sends the query by itself.

In case 2, peer x drops the query. If no relay from the next hop is received in a given timeout, initiator will determine that x drops it.

In case 3, peer x returns initiator an incorrect next hop. If the incorrect next hop colludes, the fact that initiate cannot verify the identity of the next hop makes determining which peer Sends the incorrect reply impossible. This challenge causes us to believe that the technique of verifying the ID of remote peer is

necessary. Combined with Peer-ID based signature scheme, initiator can identify the malicious node x.

2.3 PROPOSED SYSTEM

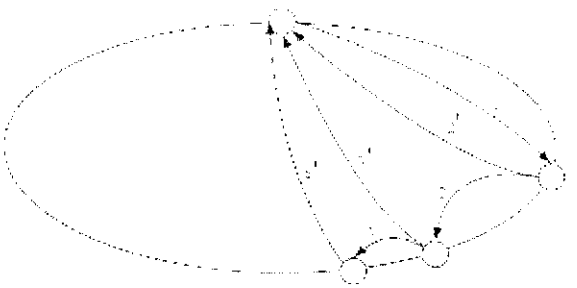
2.3.1 Tracer routing

To make the routing strategy perform best, we present an efficient routing strategy, called tracer routing. Tracer routing enables the initiator to trace the whole routing process. It can reduce *normal* routing latency from $2h * t$ to $(h + 1) * t$. The basic principle of the routing strategy is as follows. In each step, the intermediate peer x not only forwards the query to the next hop, but also returns the IP address of the next hop to initiator. With the additional information, the initiator has the knowledge about the whole routing process. Each intermediate peer directly forwards the query to the next hop, thus the query can be routed quickly. Combined with the peer-ID based signature scheme, tracer routing offers a good tradeoff between routing efficiency and security.

We propose to address routing message attack by combined tracer routing with Peer-ID based signature scheme. Note that Peer-ID based signature scheme is not necessary. Any techniques of verifying the Peer-ID of remote peer can work with tracer routing. In our scheme, the initiator appends a signature to a query. When a intermediate peer x receives

The message (including query and its signature), x verifies the message and discards the polluted or forged one using the initiator's public key. Recall that the public key is the Peer- ID of initiator. Then x forwards the message it received to the next hop. At the same time, x sends an acknowledgement (including the Peer-

ID of the next hop, query and the signature generated using the private key of x) to initiator. The process is repeated until the query reaches the target.



Tracer Routing

O - Node 2',3',4'- ACK & Next IP address
1,2,3-Encrypted Message & Destination IP address

Message routing: At each routing step, a node seeks to forward the message to a node in the routing table whose node Id shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current node's id. If no such node can be found, the message is forwarded to a node whose node Id shares a prefix with the key as long as the current node, but is numerically closer to the key than the current node's id. If no appropriate node exists in either the routing table or neighbor set, then the current node or its immediate neighbor is the message's final destination.

P2P systems rely on other peers for message routing, thus each message should be properly forwarded to the next hop without any modification. Malicious nodes may attack the message routing in the following ways:

1. alter the message in transit.

2. disrupt the message routing, or take advantage of locality

to control some routes.

3. pretend to be the target

In order to prevent attacks mentioned above, our scheme should contain sufficient protection against attack manipulating the overlay routing. The authenticity and integrity of messages can be handled by using signatures. A peer receiving

a message can confirm that the message is not altered by verifying the signature. The signature of initiator allows the attack to be detected, the malicious peer to be identified. The routing path of a message can be controlled by the initiator. On detecting that the message is forwarded to a wrong place, the initiator will search an alternate routing path used to circumvent malicious peers.

2.4 SYSTEM SPECIFICATION

2.4.1 HARDWARE REQUIREMENT SPECIFICATIONS:

PROCESSOR	:	PENTIUM IV
SPEED	:	ABOVE 500 MHZ
RAM CAPACITY	:	256 MB MIN
HARD DISK DRIVE	:	80 GB
MONITOR	:	15' COLOR

2.4.2 SOFTWARE REQUIREMENT SPECIFICATIONS:

OPERATING SYSTEM	:	WINDOWS 7/XP
ENVIRONMENT	:	VISUAL STUDIO .NET 2008
DOTNET FRAMEWORK	:	VERSION 3.5
LANGUAGE	:	VB.NET

2.4.3 SOFTWARE DESCRIPTION

2.4.3.1 VB.Net

VB.net, provides the enhanced functionality of the win32 api to intermediate and advanced visual basic developers using 'VB Classic'. All code is provided free of charge but, I do have simple reproduction / distribution rules that ask be adhered to. For more information, please read the VB*net* licensing policy. And remember that when you're finished at VB*net*, there are plenty of other hardcore links both at here at mvps.org, as well as on the VB*net* MVP and Best Links pages. N

Visual Basic .NET (VB.NET) is an object-oriented computer programming language that can be viewed as an evolution of Microsoft's Visual Basic (VB) which is generally implemented on the Microsoft .NET Framework. Microsoft currently supplies Visual Basic Express Edition free of charge.

Criticism:

Long-time Visual Basic users have complained [13] about Visual Basic .NET because initial versions dropped a large number of language constructs and user interface features [14] that were available in VB6 (which is no longer sold by Microsoft now), and changed the semantics of those that remained; for example, in VB.NET parameters are (by default) passed by value, not by reference. Detractors refer pejoratively to VB.NET as *Visual Fred* or *DOTNOT*.[15] On March 8, 2005, a petition [16] was set up in response to Microsoft's refusal to extend its mainstream support[17] for VB6.

VB.NET's supporters state that the new language is in most respects more powerful than the original, incorporating modern object oriented programming paradigms in a more natural, coherent and complete manner than was possible with earlier versions. Opponents tend to respond that although VB6 has flaws in its object

model, the cost in terms of redevelopment effort is too high for any benefits that might be gained by converting to VB.NET.[citation needed]

It is simpler to decompile languages that target Common Intermediate Language (CIL), including VB.NET, compared to languages that compile to machine code. Tools such as .NET Reflector can provide a close approximation to the original code due to the large amount of metadata provided in CIL.

Microsoft supplies an automated VB6-to-VB.NET converter with Visual Studio .NET, which has improved over time, but it cannot convert all code, and almost all non-trivial programs will need some manual effort to compile. Most will need a significant level of code refactoring to work optimally. Visual Basic programs that are mainly algorithmic in nature can be migrated with few difficulties; those that rely heavily on such features as database support, graphics, unmanaged operations or on implementation details are more troublesome.

Both Visual Basic 6 and Visual Basic .NET will automatically generate the Sub and End Sub statements when the corresponding button is clicked in design view. Visual Basic .NET will also generate the necessary Class and End Class statements. The developer need only add the statement to display the "Hello, World" message box.

Note that all procedure calls must be made with parentheses in VB.NET, whereas in VB6 there were different conventions for functions (parentheses required) and subs (no parentheses allowed, unless called using the keyword Call).

Also note that the names Command1 and Button1 are not obligatory. However, these are default names for a command button in VB6 and VB.NET respectively.

In VB.NET, the Handles keyword is used to make the sub Button1_Click a handler for the Click event of the object Button1. In VB6, event handler subs must have a

specific name consisting of the object's name ("Command1"), an underscore ("_"), and the event's name ("Click", hence "Command1_Click").

There is a function called MsgBox in the Microsoft.VisualBasic namespace which can be used similarly to the corresponding function in VB6. There is a controversy about which function to use as a best practice (not only restricted to showing message boxes but also regarding other features of the Microsoft.VisualBasic namespace). Some programmers prefer to do things "the .NET way", since the Framework classes have more features and are less language-specific. Others argue that using language-specific features makes code more readable (for example, using int (C#) or Integer (VB.NET) instead of System.Int32).

In VB 2008, the inclusion of ByVal sender as Object, ByVal e as EventArgs has become optional

In addition, the required runtime libraries for VB6 programs are provided with Windows 98 SE and above, while VB.NET programs require the installation of the significantly larger .NET Framework. The framework is included with Windows 7, Windows Vista, Windows XP Media Center Edition, Windows XP Tablet PC Edition, Windows Server 2008 and Windows Server 2003. For other supported operating systems such as Windows 2000 or Windows XP (Home or Professional Editions), it must be separately installed.

Microsoft's response to developer dissatisfaction has focused around making it easier to move new development and shift existing codebases from VB6 to VB.NET. Their latest offering is the VBRun website, which offers code samples and articles for:

- Using VB.NET to complete tasks that were common in VB6, like creating a print preview

- Integrating VB6 and VB.NET solutions (dubbed *VB Fusion*)

2.4.4 SECURITY DESCRIPTION

2.4.4.1DES Encryption and decryption

Encryption is a method which allows information to be hidden so that it cannot be read without special knowledge or tools. Once this is done the information is **encrypted**. **Decryption** is a way to change an encrypted piece of information back into unencrypted form. This is called the **decrypted** form.

The System.Security.Cryptographic namespace in the .NET Framework provides a variety of tools to aid in encryption and in decryption. The **CryptoStream** class is one of the many classes that is provided. The **CryptoStream** class is designed to encrypt or to decrypt content as that content is streamed out to a file.

2.4.4.2 DES ENCRYPTION TECHNIQUE

To encrypt a file, follow these steps:

1. Run Visual Studio .NET or Visual Studio 2005.
2. Create a new console application in Visual Basic .NET or in Visual Basic 2005. A module is created for you, together with an empty **Main()** procedure.
3. Use the **Imports** statement on the System namespace, the System.Security namespace, the System.Security.Cryptography namespace, the System.Text namespace, and the System.IO namespace.

You must do this so that you do not have to qualify declarations from these namespaces later in your code. You must use these statements before any other declarations.

4. Imports System

5. Imports System.IO

6. Imports System.Security

7. Imports System.Security.Cryptography

8. Imports System.Runtime.InteropServices

9. Imports System.Text

10. Generate a secret key to encrypt and to decrypt the data. The **DESCryptoServiceProvider** class is based on a symmetric encryption algorithm. The symmetric encryption requires a key and an initialization vector (IV) to encrypt the data. To decrypt the data, you must have the same key and the same IV. You must also use the same encryption algorithm. You can generate the keys by using either of the following methods:

    - **Method 1** You can prompt the user for a password. Then, use the password as the key and the IV.

11. **Method 2** When you create a new instance of the symmetric cryptographic classes, a new key and a new IV are automatically created for the session. You can use the key and the IV that are generated by the managed symmetric cryptographic classes to encrypt and to decrypt file.

For more information about how to generate and to distribute keys, visit the following Microsoft Web site or see the .NET Framework software development kit (SDK) documentation:

Add the following function to generate a new key for a session as noted in Method 2:

```
' Call this function to remove the key from memory after it is used for security.
Private Declare Sub ZeroMemory Lib "kernel32.dll" Alias "RtlZeroMemory" _
                (ByVal Destination As String, ByVal Length As Integer)

' Function to generate a key.
Function GenerateKey() As String
    ' Create an instance of Symmetric Algorithm. The key and the IV are generated automatically.
    Dim desCrypto As DESCryptoServiceProvider = DESCryptoServiceProvider.Create()

    ' Use the automatically generated key for encryption.
    Return ASCIIEncoding.ASCII.GetString(desCrypto.Key)
End Function
```

23. Create a method in your class that is named **EncryptFile**. The **EncryptFile** method must have three parameters:
    - *sInputFile*
    - *sOutputFile*
    - *sKey* (This is the secret key that is used to encrypt and to decrypt the file.)

```
24. Sub EncryptFile(ByVal sInputFilename As String, _
25.                 ByVal sOutputFilename As String, _
26.                 ByVal sKey As String)
27. End Sub
```

28. In the **EncryptFile** procedure, create an input **FileStream** object and an output **FileStream** object. These objects can be read from and written to the target files.

```
29.     Dim fsInput As New FileStream(sInputFilename, _
30.                                   FileMode.Open, FileAccess.Read)
31.     Dim fsEncrypted As New FileStream(sOutputFilename, _
32.                                   FileMode.Create, FileAccess.Write)
```

33. Declare an instance of the **DESCryptoServiceProvider** class. This represents the actual encryption and the actual decryption technology that is used on the files. At this point, you can create a different provider if you want to use RSA security or another cryptographic technique.

```
34. Dim DES As New DESCryptoServiceProvider()
```

35. The cryptographic provider must be provided with your secret key as an array of bytes. The System.Text namespace provides a function that is named **GetBytes()**. As part of its encoding features, the **GetBytes()** function takes a string and then returns an array of bytes. The size of the key is different for each cryptographic technique. For example, Data Encryption Standard (DES) takes a 64-bit key that is equal to 8 bytes or to 8 characters.

    If you do not provide a key, the provider randomly generates one. This successfully encrypts the file, but there is no way to decrypt the file. Note that you also must provide the IV. This value is used as part of the encryption. Like the key, the IV is randomly generated if you do not provide the value. Because the values must be the same for both the encryption and the decryption, you must not permit random generation of these values.

36. DES.Key = ASCIIEncoding.ASCII.GetBytes(sKey)

37. DES.IV = ASCIIEncoding.ASCII.GetBytes(sKey)

38. Create an instance of the **CryptoStream** class. Use the cryptographic provider to obtain an encrypting object (**CreateEncryptor**) and the existing output **FileStream** object as a part of the constructor.

39. Dim cryptostream As New CryptoStream(fsEncrypted, _

40. desencrypt, _

41. CryptoStreamMode.Write)

42. Read in the input file, and then write out to the output file. Pass through the **CryptoStream** object where the file is encrypted by using the key that you provided.

43. Dim bytearrayinput(fsInput.Length - 1) As Byte

44. fsInput.Read(bytearrayinput, 0, bytearrayinput.Length)

cryptostream.Write(bytearrayinput, 0, bytearrayinput.Length)

2.4.4.3 DES DECRYPTION TECHNIQUE

To decrypt a file, follow these steps:

1. Create a method that is named **DecryptFile**. The decryption process is similar to the encryption process. However, **DecryptFile** has two key differences from the **EncryptFile** procedure.
    - **CreateDecryptor** is used instead of **CreateEncryptor** to create the **CryptoStream** object that specifies how the object can be used.
    - When the decrypted text is written to the destination file, the **CryptoStream** object is now the source instead of the destination stream.

2. Sub DecryptFile(ByVal sInputFilename As String, _
3. ByVal sOutputFilename As String, _
4. ByVal sKey As String)
5. Dim DES As New DESCryptoServiceProvider()

```
6.  'A 64-bit key and an IV are required for this provider.
7.  'Set secret key for DES algorithm.
8.  DES.Key() = ASCIIEncoding.ASCII.GetBytes(sKey)
9.  'Set initialization vector.
10.  DES.IV = ASCIIEncoding.ASCII.GetBytes(sKey)
11.
12.  'Create a file stream to read the encrypted file back.
13.  Dim fsread As New FileStream(sInputFilename, FileMode.Open,
     FileAccess.Read)
14.  'Create a DES Decryptor from your DES instance.
15.  Dim desdecrypt As ICryptoTransform = DES.CreateDecryptor()
16.  'Create a crypto stream set to read and to do a DES decryption
     transform on incoming bytes.
17.  Dim cryptostreamDecr As New CryptoStream(fsread, desdecrypt,
     CryptoStreamMode.Read)
18.  'Print out the contents of the decrypted file.
19.  Dim fsDecrypted As New StreamWriter(sOutputFilename)
20.  fsDecrypted.Write(New
     StreamReader(cryptostreamDecr).ReadToEnd)
21.  fsDecrypted.Flush()
22.  fsDecrypted.Close()
```

23. End Sub

24. Add the following lines to the **Main()** procedure to call both **EncryptFile** and **DecryptFile**.

```
25.  Public Sub Main()     'Must be 64 bits, 8 bytes.
26.     Dim sSecretKey As String
27.     ' Get the key for the file to encrypt.
28.     ' You can distribute this key to the user who will decrypt the file.
29.     sSecretKey = GenerateKey()
30.     ' For additional security, pin the key.
31.     Dim gch As GCHandle = GCHandle.Alloc(sSecretKey,
    GCHandleType.Pinned)
32.     ' Encrypt the file.
33.     EncryptFile("%USERPROFILE%\MyData.txt", _
34.                 "%USERPROFILE%\Encrypted.txt", _
35.                 sSecretKey)
36.     ' Decrypt the file.
37.     DecryptFile("%USERPROFILE%\Encrypted.txt", _
38.                 "%USERPROFILE%\Decrypted.txt", _
39.                 sSecretKey)
40.     ' Remove the key from memory.
41.     ZeroMemory(gch.AddrOfPinnedObject(), sSecretKey.Length * 2)
```

3.DESIGN AND DEVELOPMENT

42. gch.Free()

End Sub

43. Save the file. Run your application. Make sure that the path that is used for the input file name points to an existing and a not important file.

3 DESIGN AND DEVELOPMENT

3.1 SYSTEM DESIGN

System design is described as a process of planning a new business system or more to replace or to compliment an existing system. The system design states how a system will meet the requirements identified during the system analysis

System design focuses mainly on four distinct attributes. They are data structure, software architecture, interface representation and algorithmic details.

It describes a solution of approaching to the creation of new system. System design is a transmission from a user oriented document to a document oriented to programmers. It goes through a logical and physical design. The key points followed at the times of designing are:

- Preparing input and output specification
- Data flows and stores
- Preparing security and control specification
- Temporary and permanent collection of data
- A walk through before implementation
- Process

Reviewing the study phase activities and making decisions about which function are to be performed by the hardware, software, and human ware started in the design phase. The output, input and file design for each of the programs was done. Finally the generalized systems were explained to the management for approval.

The steps involved in designing phase were:

- The function to be performed is identified
- The input, output and file design is performed
- The system and component cost requirements is specified
- The design phase report is generated.

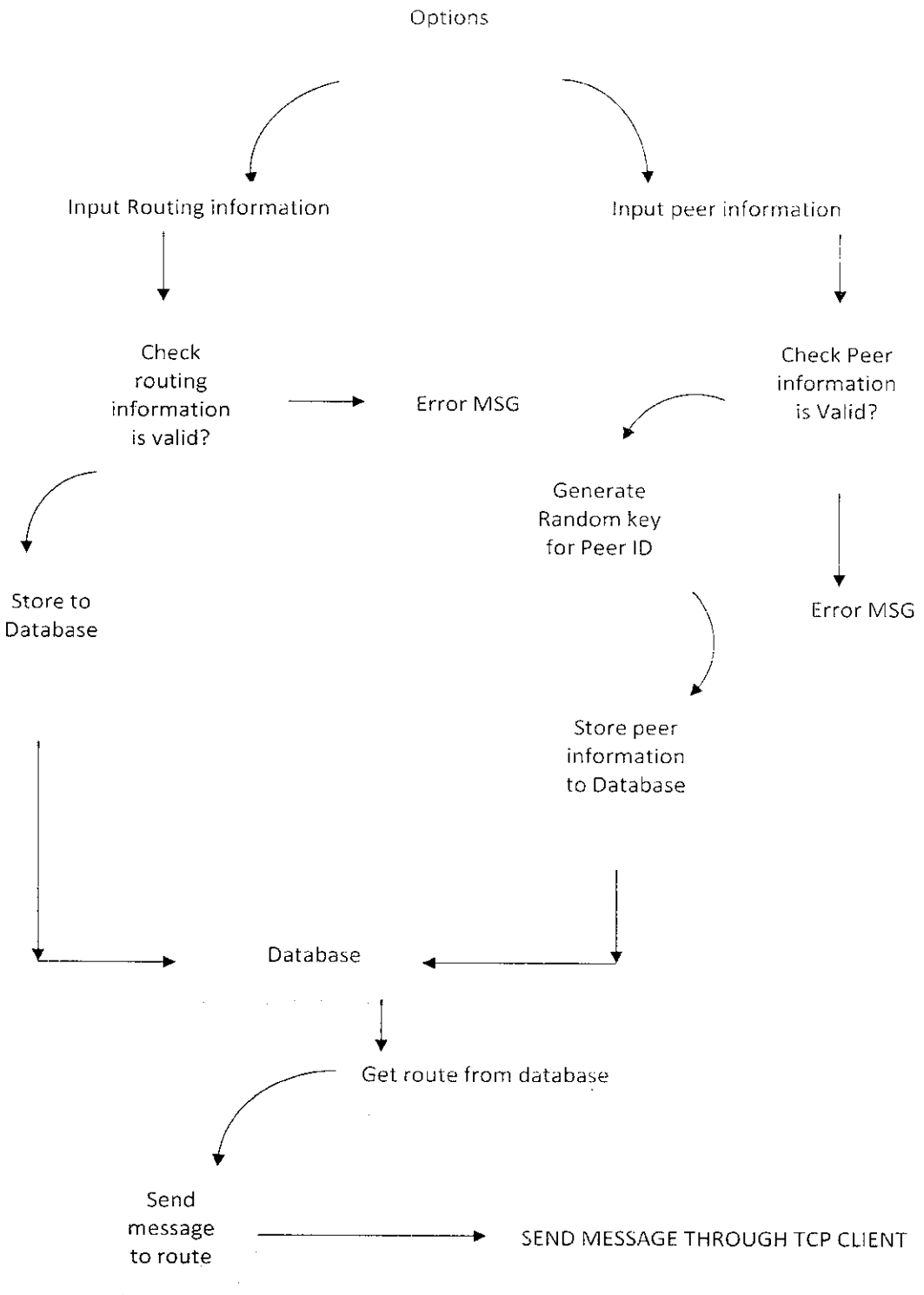
3.2 A FLOW DIAGRAM

A data flow diagram also known as "**bubble chart**" has the purpose of clarifying system requirements and identifying major transformation that will become program in system design. So it is the starting point of the phase that functionally decomposes the requirement specification down to the lowest level of details. A **DFD** contains series of bubbles joined by lines.

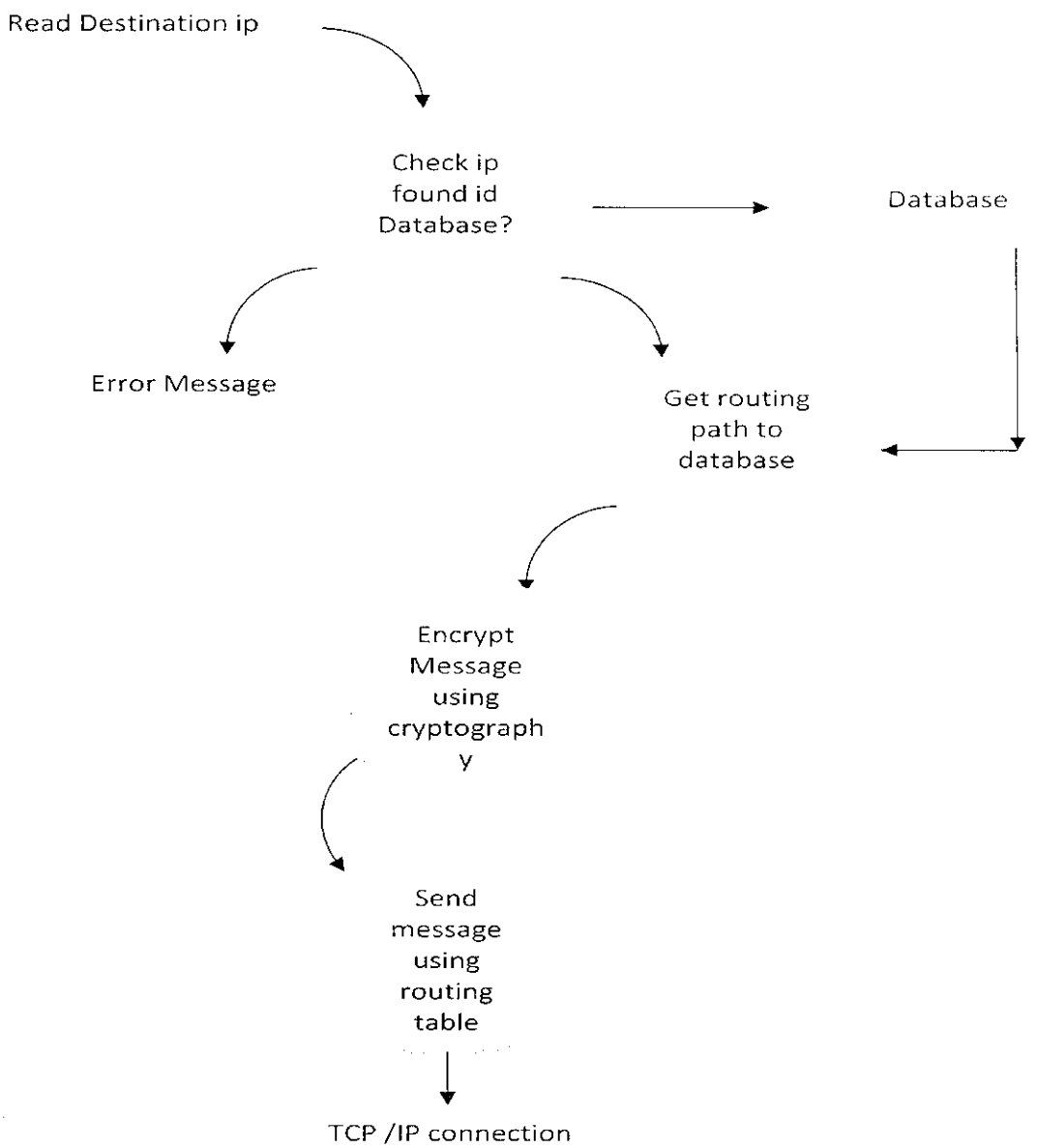
DFD SYMBOLS

- ❖ A square defines a sources or destination of system data
- ❖ An arrow identified data in motion. It is a pipeline. It is a pipeline through which information flows

SECURE MESSAGE TRANSFER PROCESS



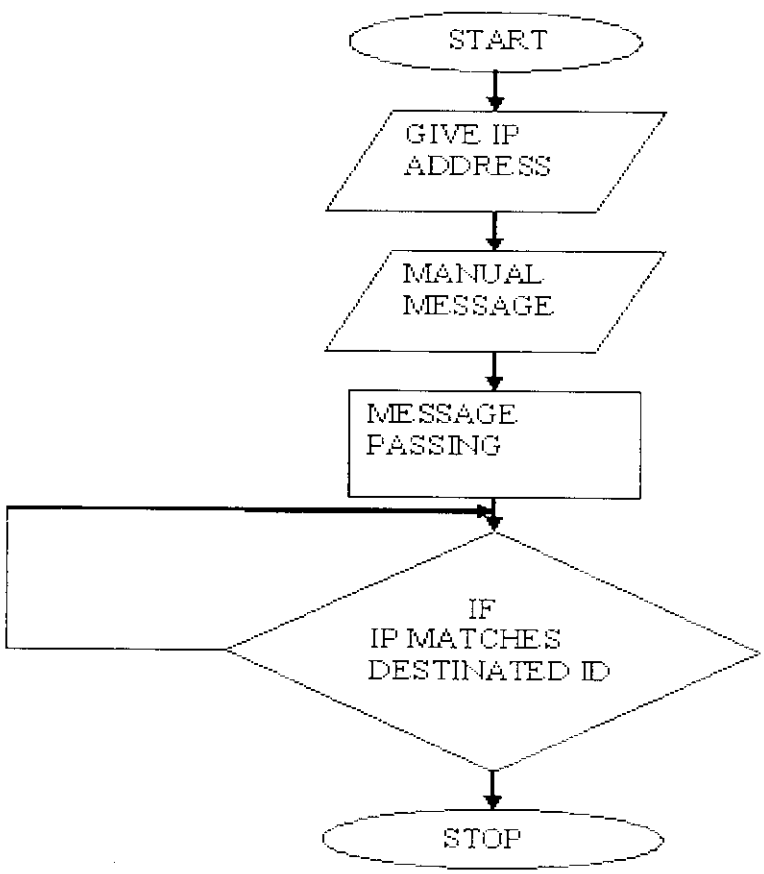
MESSAGE ENCRYPTION



System Flow Diagram

The flow of the control along with the architecture of the system is shown in the figure below. Initially, we start the message routing process by specifying the destination IP address along with the Port number. Then the message to be sent is encrypted along with its destination IP address. At the destination, the incoming message with its IP address is verified with its own IP. If it matches, then the original message is obtained through decrypting the message.

I



3.3 DESIGN PROCESS

3.3.1 INPUT DESIGN

The input design is the process of entering data to the system. The input design goal is to enter to the computer as accurate as possible. Here inputs are designed effectively so that errors made by the operations are minimized. The inputs to the system have been designed in such a way that manual forms and the inputs are coordinate where the data elements are common to the source document and to the input. The input is acceptable and understandable by the users who are using it.

The quality of the system input determines the quality for system output. Input specification describes the manner in which data entered the system processing.

Input design is the process of converting user-originated inputs to a computer-based format input data are collected and organized into group of similar data. Once identified, appropriate input media are selected for processing.

The input design also determines the user to interact efficiently with the system. Input design is a part of overall system design that requires special attention because it is the common source for data processing error. The goal of designing input data is to make entry easy and free from errors.

Modules

1. Router Option
2. Encryption and decryption

Router Option

Router receive packet message from source and check packet is own or other, if own means decrypt message using peer id else forward using routing table. Finally send acknowledgement source node.

Encryption & Decryption coding

Test this code with a text (.txt) file to confirm that the file is correctly encrypted and decrypted. Make sure that you decrypt the file to a new file (as in the **Sub Main()** procedure in this article) instead of to the original file. Examine the decrypted file and compare the decrypted file to the original.

```
Imports System
Imports System.IO
Imports System.Security
Imports System.Security.Cryptography
Imports System.Runtime.InteropServices
Imports System.Text
Module Module1
  ' Call this function to remove the key from memory after it is used for security.
  <DllImport("kernel32.dll")> _
  Public Sub ZeroMemory(ByVal addr As IntPtr, ByVal size As Integer)
  End Sub
  ' Function to generate a 64-bit key.
```

```
  Function GenerateKey() As String
    ' Create an instance of a symmetric algorithm. The key and the IV are
generated automatically.
    Dim desCrypto As DESCryptoServiceProvider =
DESCryptoServiceProvider.Create()
    ' Use the automatically generated key for encryption.
    Return ASCIIEncoding.ASCII.GetString(desCrypto.Key)
  End Function
  Sub EncryptFile(ByVal sInputFilename As String, _
            ByVal sOutputFilename As String, _
            ByVal sKey As String)
    Dim fsInput As New FileStream(sInputFilename, _
                  FileMode.Open, FileAccess.Read)
    Dim fsEncrypted As New FileStream(sOutputFilename, _
FileMode.Create, FileAccess.Write)
    Dim DES As New DESCryptoServiceProvider()
    'Set secret key for DES algorithm.
    'A 64-bit key and an IV are required for this provider.
    DES.Key = ASCIIEncoding.ASCII.GetBytes(sKey)
    'Set the initialization vector.
    DES.IV = ASCIIEncoding.ASCII.GetBytes(sKey)
```

```
    'Create the DES encryptor from this instance.
    Dim desencrypt As ICryptoTransform = DES.CreateEncryptor()
    'Create the crypto stream that transforms the file stream by using DES
encryption.
    Dim cryptostream As New CryptoStream(fsEncrypted, _
                                         desencrypt, _
                                         CryptoStreamMode.Write)
    'Read the file text to the byte array.
    Dim bytearrayinput(fsInput.Length - 1) As Byte
    fsInput.Read(bytearrayinput, 0, bytearrayinput.Length)
    'Write out the DES encrypted file.
    cryptostream.Write(bytearrayinput, 0, bytearrayinput.Length)
    cryptostream.Close()
  End Sub
  Sub DecryptFile(ByVal sInputFilename As String, _
     ByVal sOutputFilename As String, _
     ByVal sKey As String)
    Dim DES As New DESCryptoServiceProvider()
    'A 64-bit key and an IV are required for this provider.
    'Set the secret key for the DES algorithm.
```

```
    DES.Key() = ASCIIEncoding.ASCII.GetBytes(sKey)
    'Set the initialization vector.
    DES.IV = ASCIIEncoding.ASCII.GetBytes(sKey)
    'Create the file stream to read the encrypted file back.
    Dim fsread As New FileStream(sInputFilename, FileMode.Open,
FileAccess.Read)
    'Create the DES decryptor from the DES instance.
    Dim desdecrypt As ICryptoTransform = DES.CreateDecryptor()
    'Create the crypto stream set to read and to do a DES decryption transform on
incoming bytes.
    Dim cryptostreamDecr As New CryptoStream(fsread, desdecrypt,
CryptoStreamMode.Read)
    'Print out the contents of the decrypted file.
    Dim fsDecrypted As New StreamWriter(sOutputFilename)
    fsDecrypted.Write(New StreamReader(cryptostreamDecr).ReadToEnd)
    fsDecrypted.Flush()
    fsDecrypted.Close()
  End Sub

  Public Sub Main()
    'Must be 64 bits, 8 bytes.
```

```
    Dim sSecretKey As String
    ' Get the key for the file to encrypt.
    ' You can distribute this key to the user who will decrypt the file.
    sSecretKey = GenerateKey()
    ' For additional security, pin the key.
    Dim gch As GCHandle = GCHandle.Alloc(sSecretKey,
GCHandleType.Pinned)
    ' Encrypt the file.
    EncryptFile("%USERPROFILE%\MyData.txt", _
                "%USERPROFILE%\Encrypted.txt", _
                sSecretKey)
    ' Decrypt the file.
    DecryptFile("%USERPROFILE%\Encrypted.txt", _
                "%USERPROFILE%\Decrypted.txt", _
                sSecretKey)
    ' Remove the key from memory.
    ZeroMemory(gch.AddrOfPinnedObject(), sSecretKey.Length * 2)
    gch.Free()
  End Sub
End Module
```

4. IMPLEMENTATION

3.2.3 OUTPUT DESIGN

The output design was done so that results of processing could be communicated to the users. The various outputs have been designed in such a way that they represent the same format that the office and management used to.

Computer output is the most important and direct source of information to the user. Efficient, intelligible output design should improve the systems relationships with the user and help in decision making. A major form of output is hardcopy from the printer

Output requirements are designed during system analysis. A good starting point for the output design is the Data Flow Diagram (DFD). Human factors educe issues for design involves addressing internal controls to ensure readability

4 IMPLEMENTATION

4.1 SYSTEM IMPLEMENTATION

The purpose of System Implementation can be summarized as follows:

It making the new system available to a prepared set of users (the deployment), and positioning on-going support and maintenance of the system within the Performing Organization (the transition). At a finer level of detail, deploying the system consists of executing all steps necessary to educate the Consumers on the use of the new system, placing the newly developed system into production, confirming that all data required at the start of operations is available and accurate, and validating that business functions that interact with the system are functioning properly. Transitioning the system support responsibilities involves changing from a system development to a system support and maintenance mode of operation,

with ownership of the new system moving from the Project Team to the Performing Organization.

System implementation is the important stage of project when the theoretical design is tuned into practical system. The main stages in the implementation are as follows:

- Planning
- Training
- System testing and
- Changeover Planning

Planning is the first task in the system implementation. Planning means deciding on the method and the time scale to be adopted. At the time of implementation of any system people from different departments and system analysis involve. They are confirmed to practical problem of controlling various activities of people outside their own data processing departments. The line managers controlled through an implementation coordinating committee. The committee considers ideas, problems and complaints of user department, it must also consider;

- The implication of system environment
- Self selection and allocation form implementation tasks
- Consultation with unions and resources available
- Standby facilities and channels of communication

5. CONCLUSION

5 CONCLUSIONS

Peer-to-peer overlay networks, both at the network layer and at the application layer. We have shown how techniques ranging from cryptography through redundant routing to economic methods can be applied to increase the security, fairness, and trust for applications on the p2p network. Because of the diversity of how p2p systems are used, there will be a corresponding diversity of security solutions applied to the problems. It has presented the design and analysis of techniques for secure node joining, routing table maintenance, and message forwarding in structured p2p overlays. These techniques provide secure routing, which can be combined with existing techniques to construct applications that are robust in the presence of malicious participants.

6 APPENDICES

6.1 SOURCE CODE:

CLIENT CONNECTION:

```
Imports System.Net.Sockets

Imports System.Text

Imports System.IO

Imports System.Threading

Imports System

Imports System.Xml

Imports System.Security.Cryptography
```

6. APPENDICES

```
Imports System.Data.OleDb


Public Class ClientConnection
    Dim clientSocket As New System.Net.Sockets.TcpClient()
    Dim serverStream As NetworkStream
    Private clientStreamWriter As StreamWriter
    Private clientStreamReader As StreamReader
    Dim readData As String
    Dim infiniteCounter As Integer
    Private Shared key() As Byte = {}
    Private Shared IV() As Byte = {&H12, &H34, &H56, &H78, &H90, &HAB,
&HCD, &HEF}
    Private Const EncryptionKey As String = "abcdefgh"
    Private dataConnection As OleDbConnection
    Private Sub btnConnect_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnConnect.Click
        Dim dataReader As OleDb.OleDbDataReader
        Dim routeString As String = ""
        Dim cmdStr As String = "Select * FROM route where dest = '" +
txtIpAddress.Text + "'"
        setDBConnection()
```

```
        Dim oleCmd As OleDbCommand
        oleCmd = New OleDbCommand(cmdStr, dataConnection)
        dataReader = oleCmd.ExecuteReader()
        While (dataReader.Read())
            routeString = dataReader(2)
        End While
        readData = "Conected to Chat Server ..."
        Dim nextRoute As String() = routeString.Split(New Char() {";"c})
        Dim nextIP As String() = nextRoute(0).Split(New Char() {","c})
        'MsgBox(nextIP(0))
        'MsgBox(nextIP(1))
        'clientSocket.Connect(txtIpAddress.Text, Int32.Parse(txtPort.Text))
         clientSocket.Connect(nextIP(0), Int32.Parse(nextIP(1)))
        serverStream = clientSocket.GetStream()
        clientStreamWriter = New StreamWriter(serverStream)
        clientStreamWriter.AutoFlush = True
        Dim encStr As String = EncryptAES(txtMessage.Text)
        clientStreamWriter.WriteLine("" + encStr.Length.ToString() + "?" + encStr +
":" + routeString)
        clientSocket.Close()
```

```
        Dim ctThread As Threading.Thread = New Threading.Thread(AddressOf
getMessage)
        'ctThread.Start()
    End Sub
     Private Sub setDBConnection()
        Dim conStr As String
        Dim dataPath As String = "d:/secure.mdb"
        conStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & dataPath &
";Persist Security Info=False"
        dataConnection = New OleDbConnection(conStr)
        dataConnection.Open()
    End Sub
    Private Sub getMessage()
        For infiniteCounter = 1 To 2
            infiniteCounter = 1
            serverStream = clientSocket.GetStream()
            clientStreamReader = New StreamReader(serverStream)
            Thread.Sleep(1000)
            Dim nameType As String = clientStreamReader.ReadLine()
            readData = "" + nameType
            msg()
```

```
        Next
    End Sub
    Private Sub msg()
        If Me.InvokeRequired Then
            Me.Invoke(New MethodInvoker(AddressOf msg))
        Else
            txtMessage.Text = txtMessage.Text + Environment.NewLine + " >> " +
readData
        End If
    End Sub
    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnClose.Click
        Visible = False
    End Sub
    Public Shared Function EncryptAES(ByVal stringToEncrypt As String) As
String
        Try
            key = System.Text.Encoding.UTF8.GetBytes(EncryptionKey)
            Dim des As New DESCryptoServiceProvider
            Dim inputByteArray() As Byte =
Encoding.UTF8.GetBytes(stringToEncrypt)
```

```
            Dim ms As New MemoryStream
            Dim cs As New CryptoStream(ms, des.CreateEncryptor(key, IV),
CryptoStreamMode.Write)
            cs.Write(inputByteArray, 0, inputByteArray.Length)
            cs.FlushFinalBlock()
            Return Convert.ToBase64String(ms.ToArray())
        Catch ex As Exception
            'oops - add your exception logic
        End Try
        Return ""
    End Function
End Class
```

SERVERSIDE:

```
Imports System.Threading
Imports System.Net.Sockets
Imports System.IO
Imports System
Imports System.Xml
Imports System.Text
```

```
Imports System.Security.Cryptography
Public Class TCPServer
    Shared connection As Socket
    Shared readThread As Thread
    Shared socketStream As NetworkStream
    Private Shared writer As BinaryWriter
    Private Shared reader As BinaryReader
    Private Shared key() As Byte = {}
    Private Shared IV() As Byte = {&H12, &H34, &H56, &H78, &H90, &HAB,
&HCD, &HEF}
    Private Const EncryptionKey As String = "abcdefgh"
    Public Sub New()
        readThread = New Thread(AddressOf RunServer)
        readThread.Start()
    End Sub
    Public Shared Sub RunServer()
        Dim listener As TcpListener
        Dim counter As Integer = 1
        Try
            listener = New TcpListener(5000)
```

```
listener.Start()
While True
   Console.WriteLine("Waiting for connection")
   connection = listener.AcceptSocket()
   socketStream = New NetworkStream(connection)
   writer = New BinaryWriter(socketStream)
   reader = New BinaryReader(socketStream)
   Console.WriteLine("Connection " & counter & " received.")
   writer.Write("SERVER>>>" + "Connection successful")
   Dim theReply As String = ""
   Try
      Do
         theReply = reader.ReadString()
         Console.WriteLine(theReply)
      Loop While (theReply <> "CLIENT>>> TERMINATE" _
        AndAlso connection.Connected)
   Catch inputOutputException As IOException
      Console.WriteLine("Client application closing")
   Finally
      Console.WriteLine("User terminated connection")
```

```
            writer.Close()
            reader.Close()
            socketStream.Close()
            connection.Close()
            counter += 1
        End Try
    End While
  Catch inputOutputException As IOException
    Console.WriteLine("Server application closing")
  End Try
 End Sub
 Public Shared Function EncryptAES(ByVal stringToEncrypt As String) As
String
  Try
    key = System.Text.Encoding.UTF8.GetBytes(Left(EncryptionKey, 8))
    Dim des As New DESCryptoServiceProvider
    Dim inputByteArray() As Byte =
Encoding.UTF8.GetBytes(stringToEncrypt)
    Dim ms As New MemoryStream
    Dim cs As New CryptoStream(ms, des.CreateEncryptor(key, IV),
CryptoStreamMode.Write)
```

```
            cs.Write(inputByteArray, 0, inputByteArray.Length)
            cs.FlushFinalBlock()
            Return Convert.ToBase64String(ms.ToArray())
        Catch ex As Exception
            'oops - add your exception logic
        End Try
        Return ""
    End Function
    Public Function Decrypt(ByVal stringToDecrypt As String) As String
        Try
            Dim inputByteArray(stringToDecrypt.Length) As Byte
            key = System.Text.Encoding.UTF8.GetBytes(Left(EncryptionKey, 8))
            Dim des As New DESCryptoServiceProvider
            inputByteArray = Convert.FromBase64String(stringToDecrypt)
            Dim ms As New MemoryStream
            Dim cs As New CryptoStream(ms, des.CreateDecryptor(key, IV),
CryptoStreamMode.Write)
            cs.Write(inputByteArray, 0, inputByteArray.Length)
            cs.FlushFinalBlock()
            Dim encoding As System.Text.Encoding = System.Text.Encoding.UTF8
```

```
            Return encoding.GetString(ms.ToArray())
        Catch ex As Exception
            'oops - add your exception logic
        End Try
        Return "Asdf"
    End Function
End Class
```

MESSAGE DIALOG:

```
Public Class MessageDialog
    Public Delegate Sub SetTextCallback(ByVal s As String)
    Public Delegate Sub SetVisibleCallback(ByVal s As Boolean)
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()
    End Sub
    Private Sub MessageDialog_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    End Sub
    Private Sub btnOk_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOk.Click
```

```
    End Sub
    Public Sub setVisble(ByVal text As Boolean)
        If Me.InvokeRequired Then
            Dim d As New SetVisibleCallback(AddressOf setVisble)
            Me.Invoke(d, New Object() {text})
        Else
            Me.Visible = True
        End If
    End Sub
    Public Sub SetText(ByVal text As String)
        If Me.txtMessage.InvokeRequired Then
            Dim d As New SetTextCallback(AddressOf SetText)
            Me.Invoke(d, New Object() {text})
        Else
            Me.txtMessage.AppendText("" & Chr(13) & "" & Chr(10) & " " + text)
        End If
    End Sub
End Class
```

SECURITY:

```
Imports System

Imports System.IO

Imports System.Xml

Imports System.Text

Imports System.Security.Cryptography

Public Class Security

   Private key() As Byte = {}

   Private IV() As Byte = {&H12, &H34, &H56, &H78, &H90, &HAB, &HCD,
&HEF}

   Private Const EncryptionKey As String = "abcdefgh"

   Public Sub New()

   End Sub

   Public Function EncryptAES(ByVal stringToEncrypt As String) As String

      Try

         key = System.Text.Encoding.UTF8.GetBytes(Left(EncryptionKey, 8))

         Dim des As New DESCryptoServiceProvider

         Dim inputByteArray() As Byte =
Encoding.UTF8.GetBytes(stringToEncrypt)

         Dim ms As New MemoryStream
```

```
        Dim cs As New CryptoStream(ms, des.CreateEncryptor(key, IV),
CryptoStreamMode.Write)
        cs.Write(inputByteArray, 0, inputByteArray.Length)
        cs.FlushFinalBlock()
        Return Convert.ToBase64String(ms.ToArray())
      Catch ex As Exception
        'oops - add your exception logic
      End Try
      Return ""
    End Function
    Public Function Decrypt(ByVal stringToDecrypt As String) As String
      Try
        Dim inputByteArray(stringToDecrypt.Length) As Byte
        key = System.Text.Encoding.UTF8.GetBytes(Left(EncryptionKey, 8))
        Dim des As New DESCryptoServiceProvider
        inputByteArray = Convert.FromBase64String(stringToDecrypt)
        Dim ms As New MemoryStream
        Dim cs As New CryptoStream(ms, des.CreateDecryptor(key, IV),
CryptoStreamMode.Write)
        cs.Write(inputByteArray, 0, inputByteArray.Length)
        cs.FlushFinalBlock()
```

```
            Dim encoding As System.Text.Encoding = System.Text.Encoding.UTF8
            Return encoding.GetString(ms.ToArray())
        Catch ex As Exception
            'oops - add your exception logic
        End Try
        Return "Asdf"
    End Function
End Class
```

CONNECTION INFORMATION:

```
Public Class ConnectionInformation
    Dim str As String
    Public Delegate Sub SetTextCallback(ByVal s As String)
    Private Sub ConnectionInformation_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
    End Sub
    Public Sub SetText(ByVal text As String)

        If Me.txtMessage.InvokeRequired Then
            Dim d As New SetTextCallback(AddressOf SetText)
```

```
        Me.Invoke(d, New Object() {text})
      Else
        Me.txtMessage.AppendText("" & Chr(13) & "" & Chr(10) & " " + text)
      End If
    End Sub
End Class
```

SECURE MESSAGING:

```
Imports System.Windows.Forms
Imports System.Net.Sockets
Imports System.Net
Imports System.Threading
Imports System.IO
Imports System.Xml
Imports System.Text
Imports System.Security.Cryptography
Public Class SecureMessageMain
    Dim connectInfo As ConnectionInformation
    Dim serverStart As TcpListener
    Public tcpServer As TcpListener
```

```
Public tcpClient As TcpClient, client As TcpClient
Public startThread As Thread
Private clientStream As NetworkStream
Private clientStreamReader As StreamReader
Private clientStreamWriter As StreamWriter
Private key() As Byte = {}
Private IV() As Byte = {&H12, &H34, &H56, &H78, &H90, &HAB, &HCD,
&HEF}
Private Const EncryptionKey As String = "abcdefgh"
Dim clientSocket As New System.Net.Sockets.TcpClient()
Dim clientSocket1 As New System.Net.Sockets.TcpClient()
Dim serverStream As NetworkStream
Private clientStream1 As NetworkStream
Private clientStream2 As NetworkStream
Private clientStreamWriter1 As StreamWriter
Private clientStreamReader1 As StreamReader
Private clientStreamWriter2 As StreamWriter
Private clientStreamReader2 As StreamReader
Dim msg As New MessageDialog
Public msgHack, isShown As Boolean
```

```
    Public isMessageHack, isRouteHack, isIpHack As Boolean

    Dim localPort As Integer = 6666

    Private Sub ClientConnectionToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ClientConnectionToolStripMenuItem.Click

        Dim clientDetails As New ClientConnection()

        clientDetails.MdiParent = Me

        clientDetails.Show()

    End Sub

    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click

        End

    End Sub

    Private Sub IPConfigToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
IPConfigToolStripMenuItem.Click

        Dim routeConfig As New RouteConfig()

        routeConfig.ShowDialog()

    End Sub

    Private Sub SecureMessageMain_Load(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles MyBase.Load

        localPort = InputBox("Enter Port")
```

```
        connectInfo = New ConnectionInformation()
        connectInfo.MdiParent = Me
        connectInfo.SetBounds(750, 10, 260, 600)
        connectInfo.Show()
        connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Server Started Port at "
+ localPort.ToString())
        connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Server is Waiting for
Client Request.....")
        msg.MdiParent = Me
        StartServer()
    End Sub
    Public Sub StartServer()
        Dim startThread As Thread
        startThread = New Thread(New ThreadStart(AddressOf StartListen))
        startThread.Start()
    End Sub
    Public Sub StartListen()
        Try
            tcpServer = New TcpListener(localPort)
            tcpServer.Start()
            While True
```

```
            tcpClient = tcpServer.AcceptTcpClient()
            Dim remoteIP As String = DirectCast(tcpClient.Client.RemoteEndPoint,
IPEndPoint).Address.ToString()
            Dim remotePort As String =
DirectCast(tcpClient.Client.RemoteEndPoint, IPEndPoint).Port.ToString()
            connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "New Connection
Accepted  IP @" + remoteIP)
            connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Connected Port
@" + remotePort)
            clientStream = tcpClient.GetStream()
            clientStreamReader = New StreamReader(clientStream)
            clientStreamWriter = New StreamWriter(clientStream)
            clientStreamWriter.AutoFlush = True
            Dim nameType As String = clientStreamReader.ReadLine()
            Thread.Sleep(1000)
            'MsgBox("ACK Knowledge Received From " + nameType)
            If nameType.Contains("ACK") Then
                connectInfo.SetText("ACK" + nameType)
                'connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Received From
Original  @" + (nameType))
            Else
```

```
Dim decryptStr As String() = nameType.Split(New Char() {":"c})
Dim spltstr As String = decryptStr(0).Split(New Char() {"?"c})(1)
'MsgBox("Split Value : " + spltstr)
Dim hop As String = decryptStr(1)
Dim hopAddress As String() = hop.Split(New Char() {";"c})
Dim hostString As String = ""
Dim i As Integer
For i = 1 To hopAddress.Length - 1
   hostString = hostString + hopAddress(i) + ";"
Next i
If (hostString.Length > 0) Then
   hostString = hostString.Substring(0, hostString.Length() - 1)
End If
Dim ipAddress As String = hopAddress(hopAddress.Length -
1).Split(New Char() {","c})(0)
Dim port As Integer = Int32.Parse(hopAddress(hopAddress.Length -
1).Split(New Char() {","c})(1))
Dim oAddr As System.Net.IPAddress
Dim sAddr As String
Dim hostName As String = Dns.GetHostName()
'MsgBox((nameType))
```

```
            connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Received From
Original @" + (nameType))
            With System.Net.Dns.GetHostByName(hostName)
               oAddr = New System.Net.IPAddress(.AddressList(0).Address)
               sAddr = oAddr.ToString
               sAddr.IndexOf("asdf")
            End With
            sAddr = oAddr.ToString
            If (sAddr.Equals(ipAddress) And localPort = port) Then
               Dim oLen As String = (decryptStr(0).Split(New Char()
{"?"c}))(1).Length.ToString()
               Dim cLen As String = (decryptStr(0).Split(New Char() {"?"c}))(0)
               'MsgBox("Message Length  : " + oLen)
               'MsgBox("Original Length : " + cLen)
               If (oLen.Equals(cLen)) Then
                  MsgBox("Original Message Hacking Not Found")
               Else
                  msgHack = True
                  MsgBox("Message Hacking Found")
               End If
               Dim decryptMSG As String = Decrypt(spltstr.Trim())
```

```
            connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Decrypted
@" + decryptMSG)
            connectInfo.SetText("" & Chr(13) & "" & Chr(10) & "Decrypted
@" + decryptStr(1))

            clientSocket1.Connect("192.168.1.30", Int32.Parse("5555"))
            clientStream2 = clientSocket1.GetStream()
            clientStreamReader2 = New StreamReader(clientStream2)
            clientStreamWriter2 = New StreamWriter(clientStream2)
            clientStreamWriter2.AutoFlush = True
            If (msgHack = True) Then
               clientStreamWriter2.WriteLine("ACK MESSAGE HACKED
AND REACHED DESTINATION " + Dns.GetHostName())
               clientSocket.Close()
            Else
               clientStreamWriter2.WriteLine("ACK MESSAGE REACHED
DESTINATION " + Dns.GetHostName())
               clientSocket.Close()
            End If
            msg.SetText(decryptMSG)
            If (isShown = False) Then
```

```
                isShown = True
                msg.setVisble(True)
            End If
            ' clientStreamWriter.WriteLine("Welcome I Connected")
        Else
            Dim ipPort As String() = hopAddress(1).Split(New Char() {","c})
            clientSocket1.Connect("192.168.1.30", Int32.Parse(5555))
            clientStream2 = clientSocket1.GetStream()
            clientStreamReader2 = New StreamReader(clientStream2)
            clientStreamWriter2 = New StreamWriter(clientStream2)
            clientStreamWriter2.AutoFlush = True
            clientStreamWriter2.WriteLine("ACK FROM" +
Dns.GetHostName())
            clientSocket1.Close()
            clientSocket.Connect(ipPort(0).Trim(),
Int32.Parse(ipPort(1).Trim()))
            clientStream1 = clientSocket.GetStream()
            clientStreamReader1 = New StreamReader(clientStream1)
            clientStreamWriter1 = New StreamWriter(clientStream1)
            clientStreamWriter1.AutoFlush = True
            If isMessageHack = True Then
```

```
                clientStreamWriter1.WriteLine(decryptStr(0) + "adsf" + ":" +
hostString)
                Else
                clientStreamWriter1.WriteLine(decryptStr(0) + ":" + hostString)
                End If
                clientSocket.Close()
            End If
            End If
        End While
      Catch exe As Exception
        MsgBox(exe.ToString(), MsgBoxStyle.DefaultButton1, "Error")
      End Try
    End Sub
    Sub ClientConnectionToolStripMenuItemClick(ByVal sender As Object, ByVal
e As EventArgs)
     End Sub
    Public Function Decrypt(ByVal stringToDecrypt As String) As String
      Try
        Dim inputByteArray(stringToDecrypt.Length) As Byte
        key = System.Text.Encoding.UTF8.GetBytes(EncryptionKey)
        Dim des As New DESCryptoServiceProvider
```

```
            inputByteArray = Convert.FromBase64String(stringToDecrypt)
            Dim ms As New MemoryStream
            Dim cs As New CryptoStream(ms, des.CreateDecryptor(key, IV),
CryptoStreamMode.Write)
            cs.Write(inputByteArray, 0, inputByteArray.Length)
            cs.FlushFinalBlock()
            Dim encoding As System.Text.Encoding = System.Text.Encoding.UTF8
            Return encoding.GetString(ms.ToArray())
        Catch ex As Exception
            MsgBox(ex.ToString())
        End Try
        Return "Asdf"
    End Function
    Private Sub AllowRouteHackingToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AllowRouteHackingToolStripMenuItem.Click
        isRouteHack = True
    End Sub
    Private Sub AllowIPSpoofingToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
AllowIPSpoofingToolStripMenuItem.Click
```

```
        isIpHack = True
    End Sub
    Private Sub MessageHackingToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
MessageHackingToolStripMenuItem.Click
        isMessageHack = True
    End Sub
    Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        ' Determine if text has changed in the textbox by comparing to original text.
        Application.Exit()
    End Sub 'Form1_Closing
End Class
```

ROUTE CONFIGURATION:

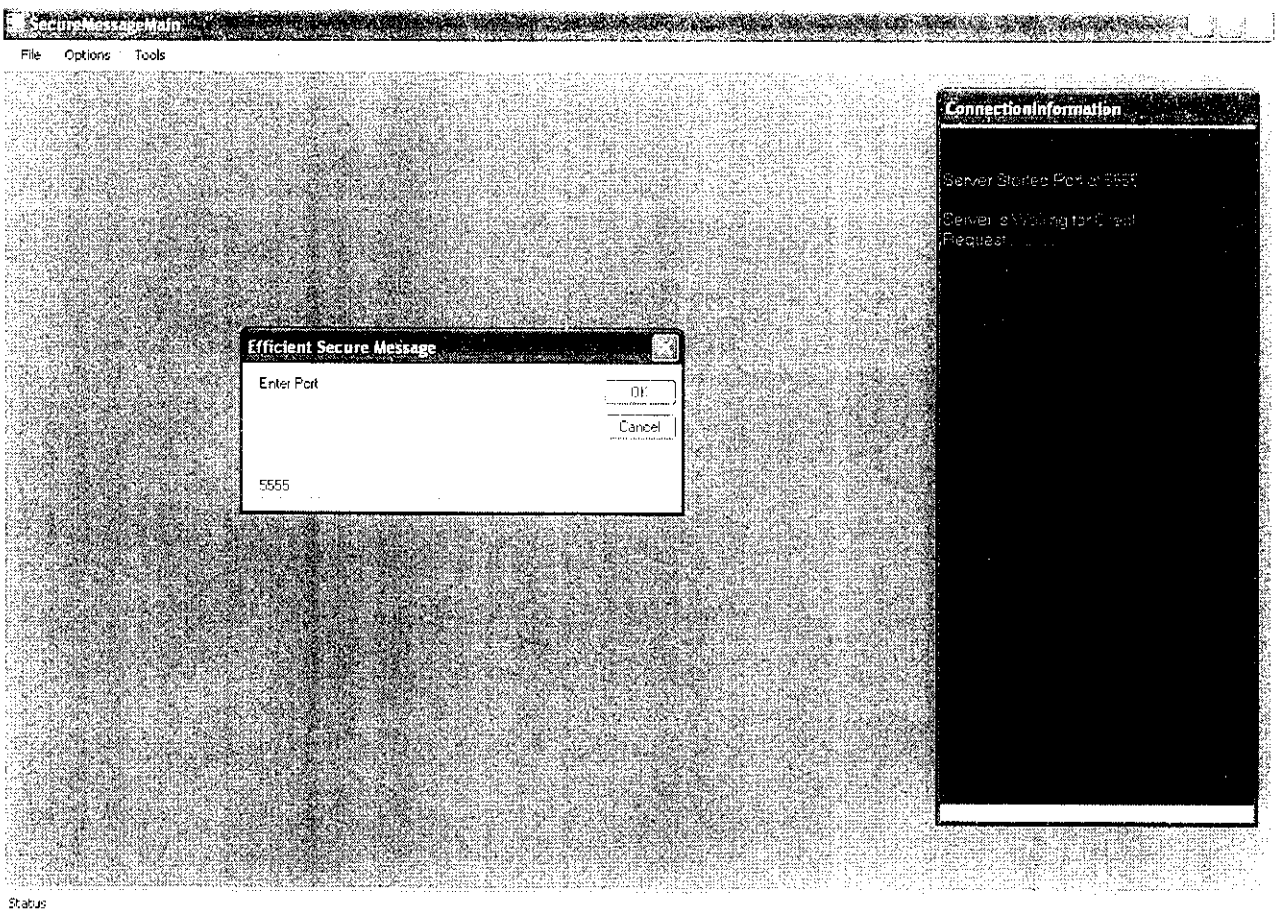
```
Imports System.Windows.Forms
Imports System.Data.OleDb
Public Class RouteConfig
    Private dataConnection As OleDbConnection
    Private Sub OK_Button_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

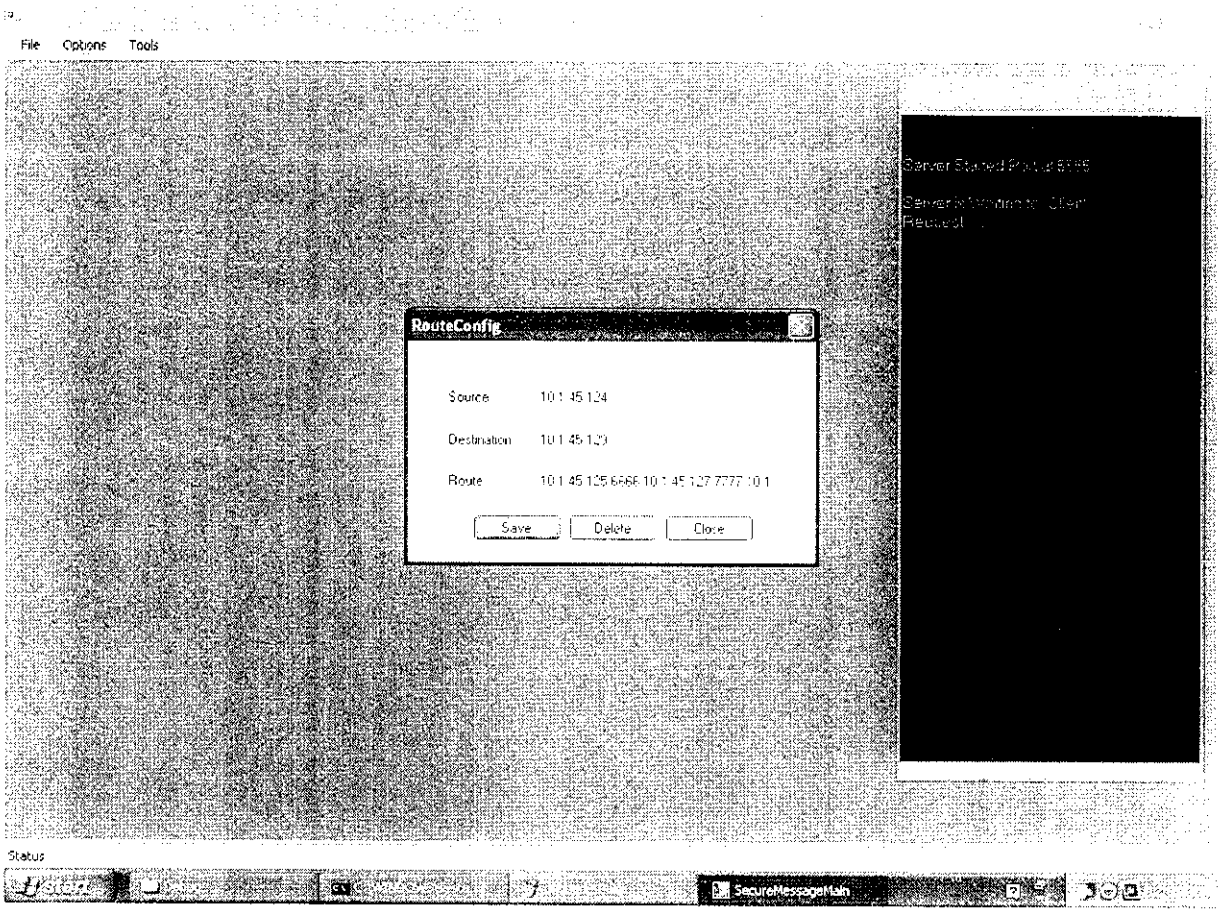
```
        Me.DialogResult = System.Windows.Forms.DialogResult.OK
        Me.Close()
    End Sub
    Private Sub Cancel_Button_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        Me.DialogResult = System.Windows.Forms.DialogResult.Cancel
        Me.Close()
    End Sub
    Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnClose.Click
        Visible = False
    End Sub
    Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSave.Click
        setDBConnection()
        Dim insertQuery As String
        Dim cmdStr As OleDbCommand
        insertQuery = "insert into route values('" + txtSource.Text + "','" +
txtDest.Text + "','" + txtRoute.Text + "')"
        cmdStr = New OleDbCommand(insertQuery, dataConnection)
        cmdStr.ExecuteNonQuery()
```

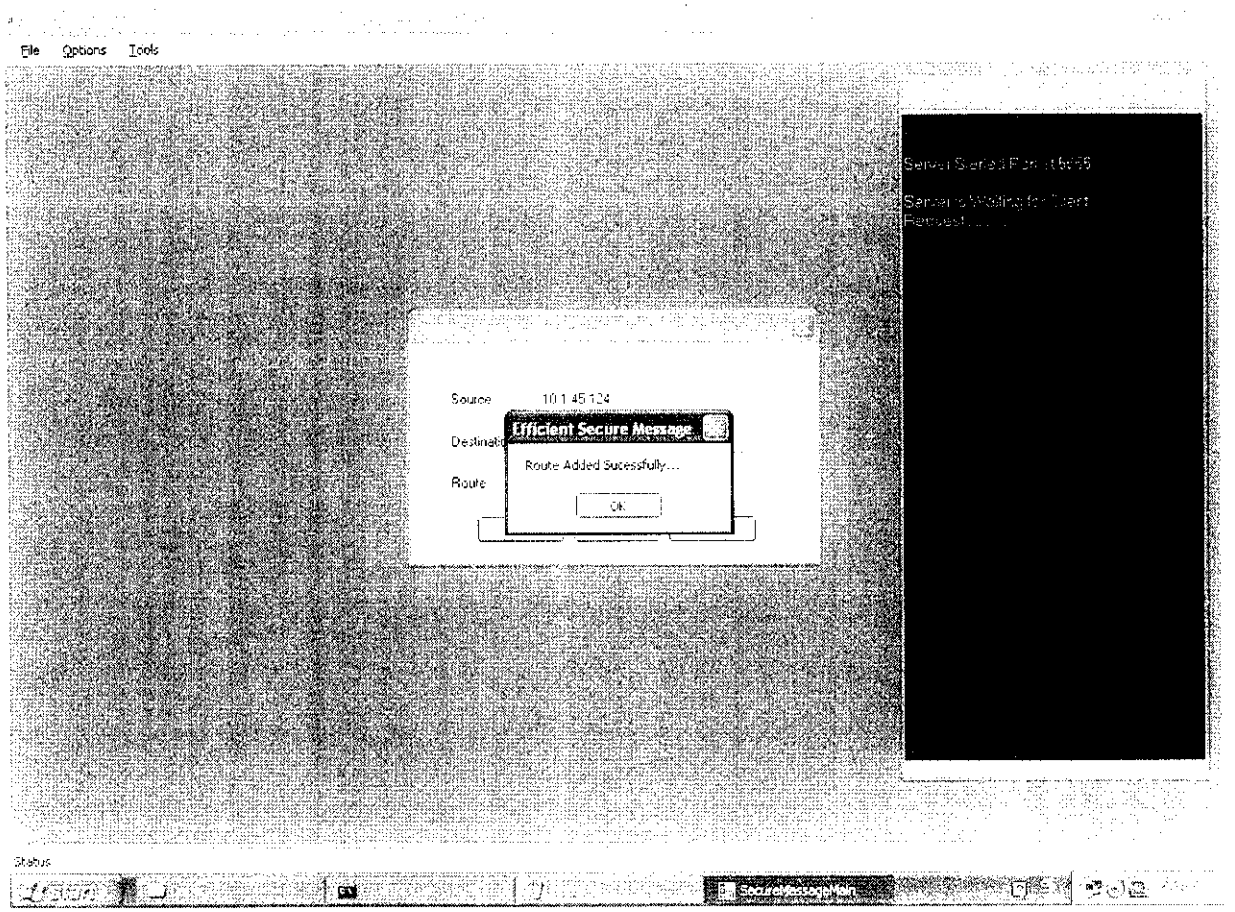
```
        dataConnection.Close()
        MsgBox("Route Added Sucessfully...")
    End Sub
    Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDelete.Click
        setDBConnection()
        Dim deleteQuery As String
        Dim cmdStr As OleDbCommand
        deleteQuery = "delete from route where source = '" + txtSource.Text + "' and
dest = '" + txtDest.Text + "'"
        cmdStr = New OleDbCommand(deleteQuery, dataConnection)
        cmdStr.ExecuteNonQuery()
        dataConnection.Close()
        MsgBox("Route Deleted Sucessfully...")
        Visible = False
    End Sub
    Private Sub setDBConnection()
        Dim conStr As String
        Dim dataPath As String = "d:/secure.mdb"
        conStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & dataPath &
";Persist Security Info=False"
```

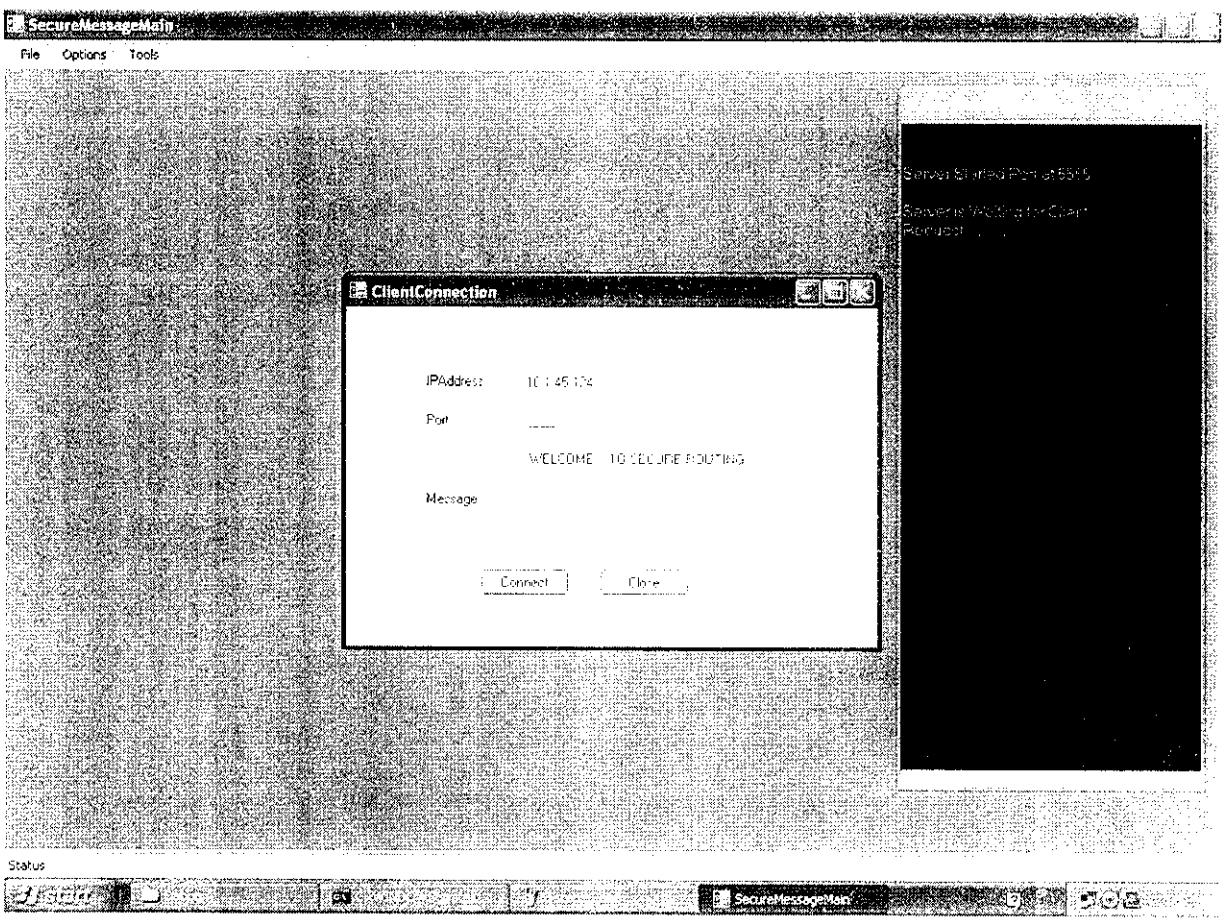
```
        dataConnection = New OleDbConnection(conStr)
        dataConnection.Open()    End Sub
    Private Sub RouteConfig_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    End Sub
End Class
```

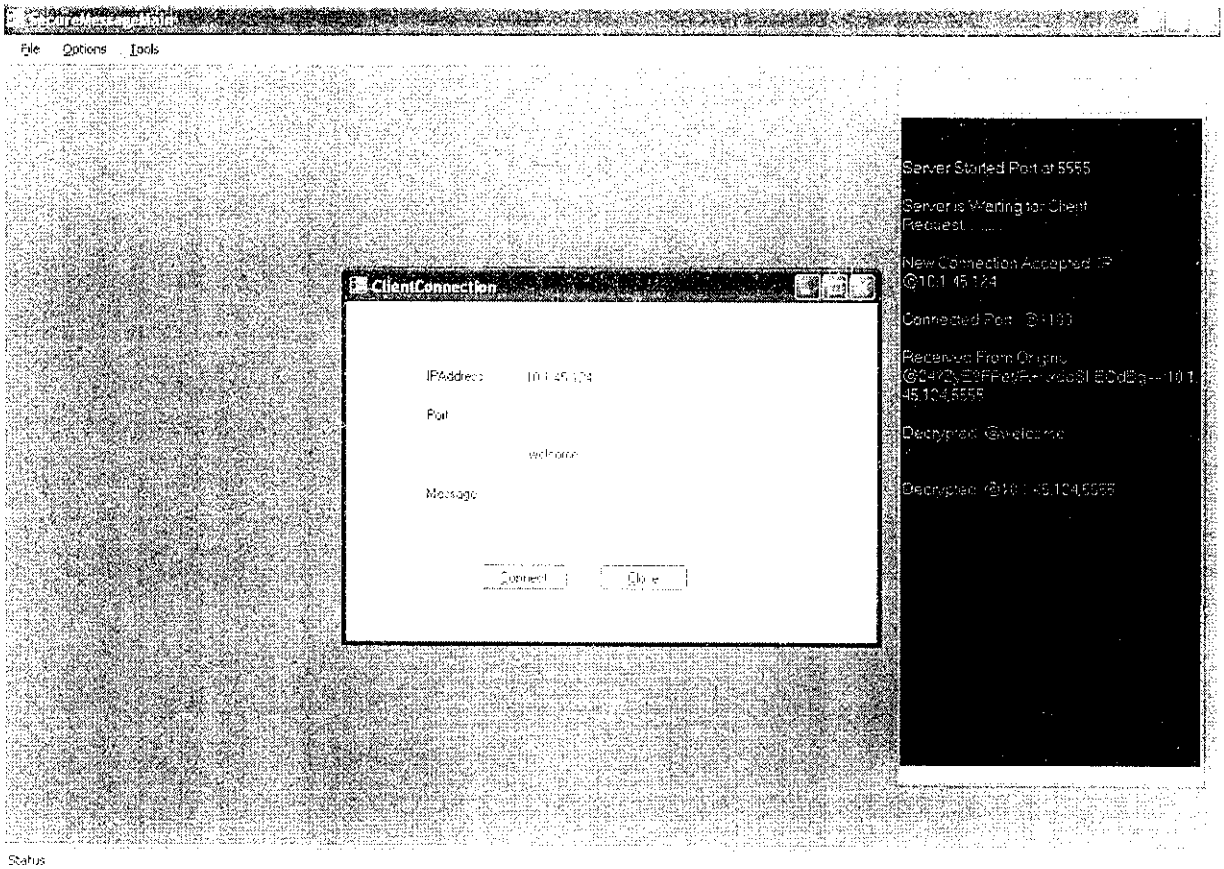
6.2 SCREEN SHOTS:

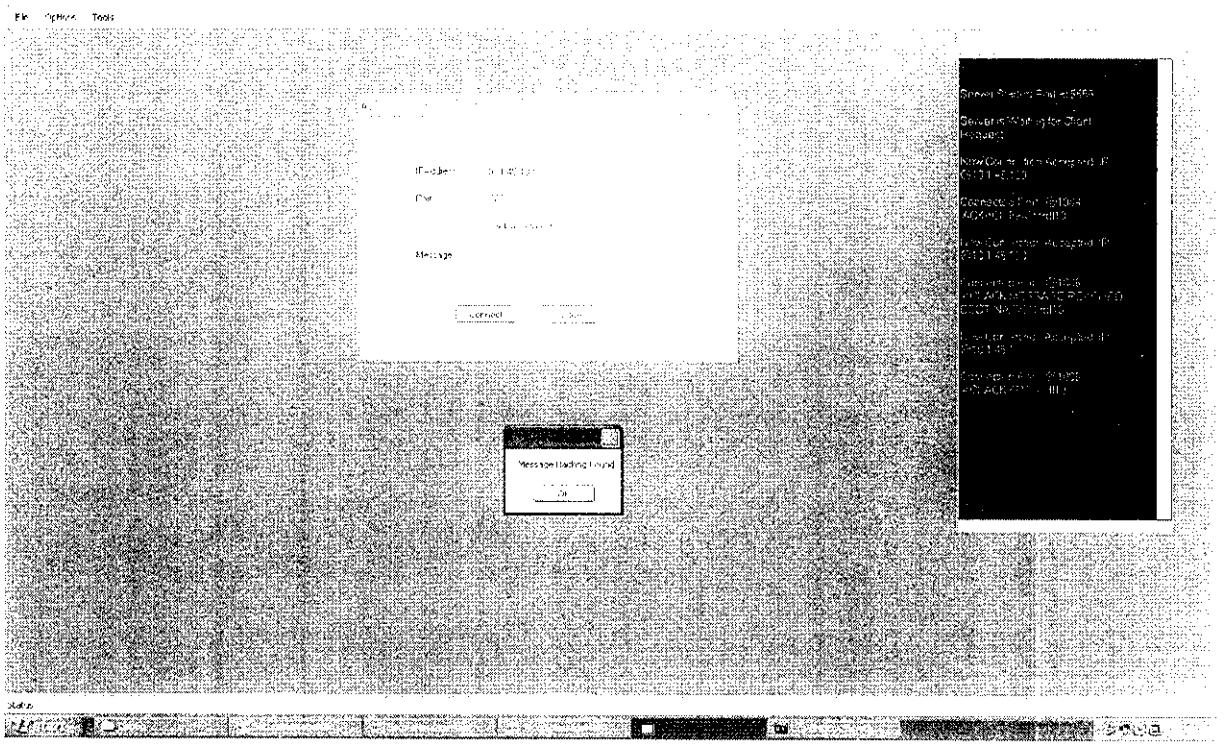


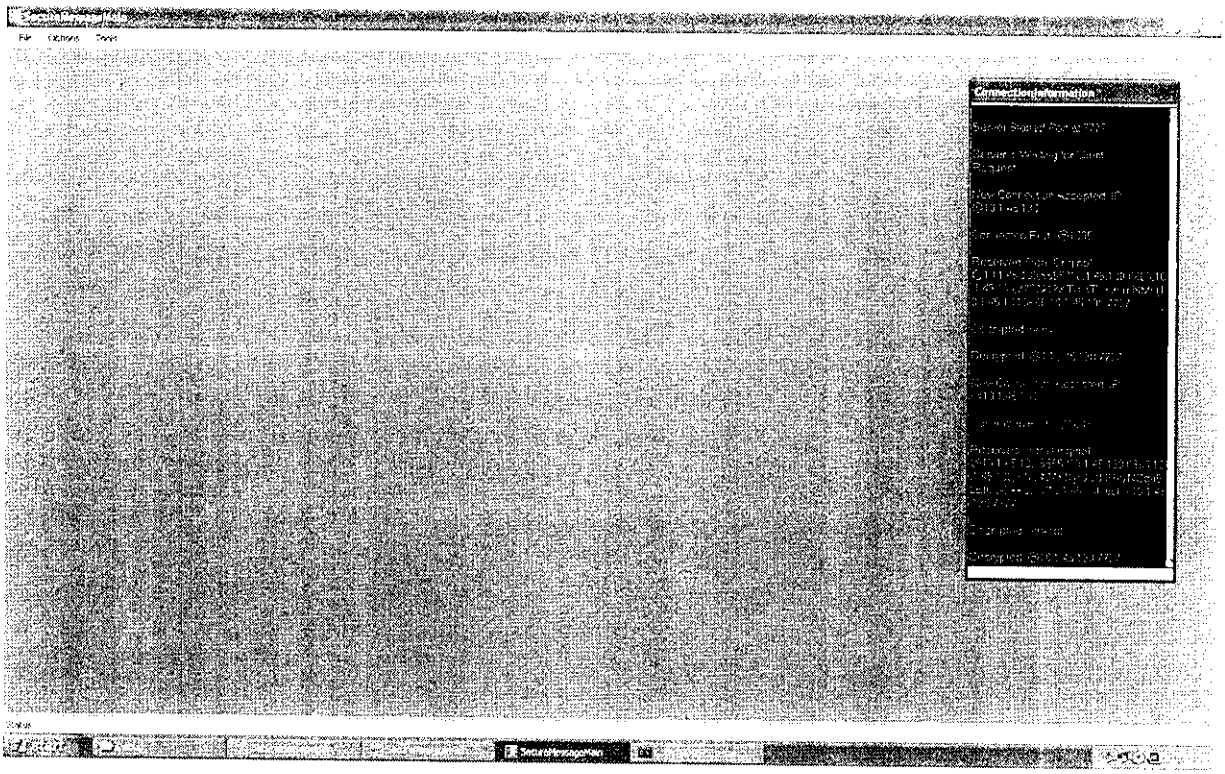

File
Options
Tools
RouteConfig
Source
Destination
Route
Save
Delete
Close
Status


File Options Tools
Source 10.1.45.124
Efficient Secure Message
Route Added Sucessfully...
OK
Route
Status


File
Options
Tools
ClientConnection
IPAddress
Port
WELCOME
Message
Connect
Close
Status


File
Options
Tools
Server Started Port at 5555
Server is Waiting for Client Request.....
New Connection Accepted IP
Connected Port
Receives From Origin
Decryption
ClientConnection
IPAddress
Port
Message
Connect
Close
Status





7. REFERENCES

6 REFERENCES:

[1] Xu Xiang, Tan Jin, Efficient Secure Message Routing for Structured Peer-to-Peer Systems, *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing-Dec 2009.*

[2] D. Wallach, A survey of peer-to-peer security issues. In *Proceedings of International Symposium of Software Security - Theories and Systems,* Nov 2003.

[3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan.

Chord: a scalable peer-to-peer lookup service for Internet applications.

In *Proceedings of SIGCOMM 2001, pages 149-160, August 2001.*

[4] A. Rowstron and P. Druschel. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science, 2218:161-172, November .*

[5] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr, 2001.

[6] J. R. Douceur, The sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer System, March 2002.*

[7] E. K. Lua, Securing peer-to-peer overlay networks from sybil attack.

In *Proceedings of International Symposium on Communications and IT, Oct 2007.*