P-3292

SECURING SOURCE ROUTING INFRASTRUCTURES

Submitted by

By

SMRUTHY BABY

Reg. No: 0820108018

of

KUMARAGURU COLLEGE OF TECHNOLOGY
(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements for the award of the degree of

MASTER OF ENGINEERING
IN

COMPUTER SCIENCE AND ENGINEERING

MAY 2010

BONAFIDE CERTIFICATE

Certified that this project report titled "**SECURING SOURCE ROUTING INFRASTRUCTURES**" is the bonafide work of **Miss. SMRUTHY BABY (0820108018)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

K. Sivan Arul Selvan

GUIDE

(Mr. K. SIVAN ARUL SELVAN)

[illegible]

HEAD OF THE DEPARTMENT

(Dr.S.THANGASAMY)

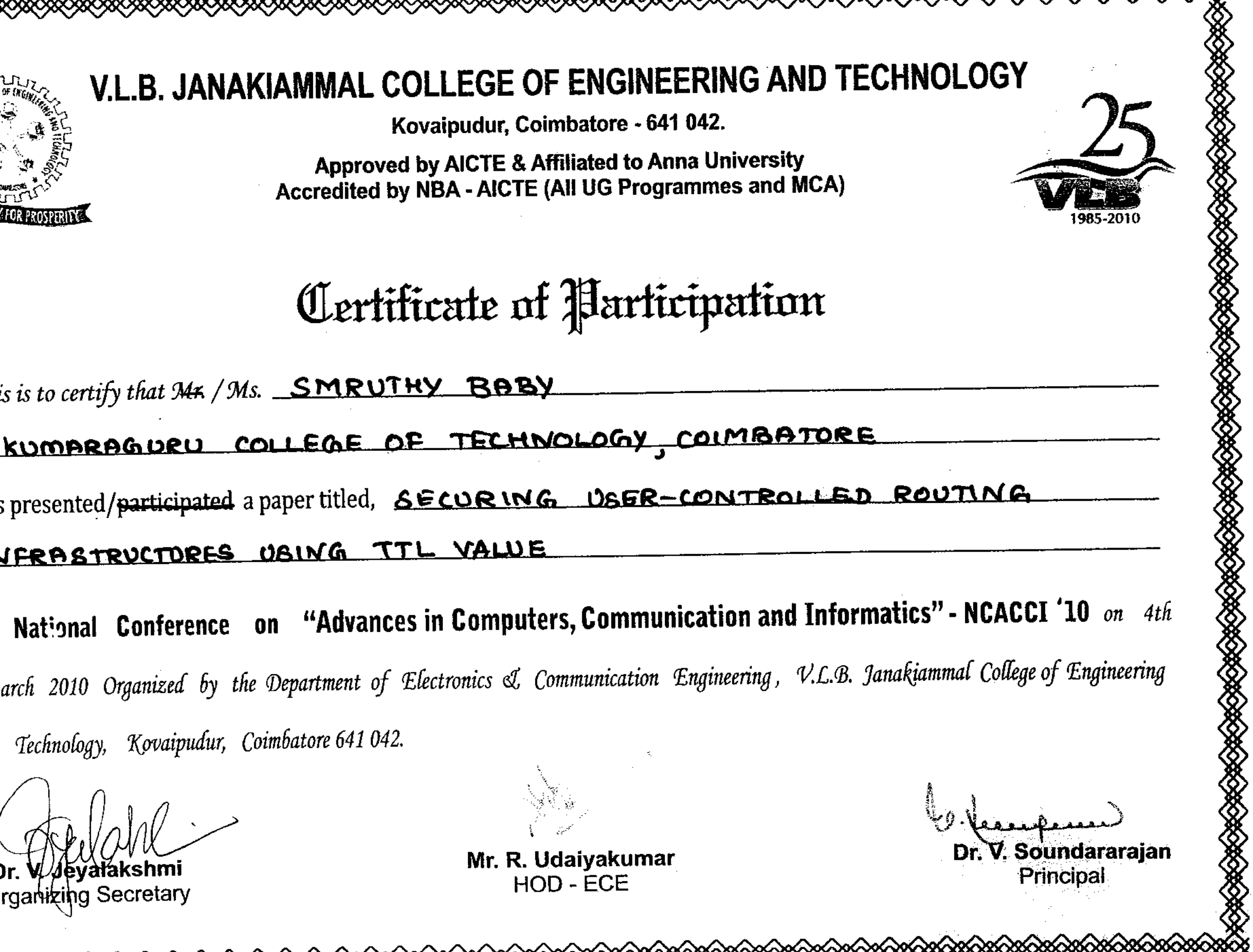
The candidate with **University Register No. 0820108018** was examined by us in Project Viva-Voce examination held on 18/5/10

[illegible] 18/5/10

Internal Examiner

[illegible]

External Examiner



V.L.B. JANAKIAMMAL COLLEGE OF ENGINEERING AND TECHNOLOGY

Kovaipudur, Coimbatore - 641 042.

Approved by AICTE & Affiliated to Anna University
Accredited by NBA - AICTE (All UG Programmes and MCA)

1985-2010

Certificate of Participation

is is to certify that ~~Mr~~ / Ms. SMRUTHY BABY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

s presented/~~participated~~ a paper titled, SECURING USER-CONTROLLED ROUTING

NFRASTRUCTURES USING TTL VALUE

National Conference on "Advances in Computers, Communication and Informatics" - NCACCI '10 *on 4th*

arch 2010 Organized by the Department of Electronics & Communication Engineering, V.L.B. Janakiammal College of Engineering

Technology, Kovaipudur, Coimbatore 641 042.

Dr. V. Jeyalakshmi
rganizing Secretary

Mr. R. Udaiyakumar
HOD - ECE

Dr. V. Soundararajan
Principal

ABSTRACT

Controlling the FI (Forwarding Infrastructures) is needed to prevent the unauthorized third parties to have control over routing infrastructures. The main technique that I introduce in this project is finding the possible paths from source to destinations using the TTL(Time To Live) value and introduces a defense mechanism based on this value so if the public key value is available, the unauthorized user can't view the message that addresses the security vulnerabilities in FI (Forwarding Infrastructure).

It is possible to prevent some of attacks like impersonation and eavesdropping on end-hosts and bound the flooding attacks that can be launched on the infrastructure nodes to a small constant value. The defense technique, which is based on lightweight cryptographic constraints on forwarding entries, prevents several attacks including eavesdropping, loops, and traffic amplification. The proposed system improves the security of the FI that provide a diverse set of operations such as packet replication. Here I consider source routing that means user can specify all possible paths from source to destinations for reaching to destination. The flexibility of forwarding infrastructures provides some security problems. These are avoided by using a defense mechanism based on TTL value. TTL value means the period of time for the messages are reaching to destination.

Here I use the defense mechanism based on symmetric cryptography--DES algorithm. These encrypted messages are forwarded through the intermediate nodes up to the destination. While reaching to destination the hop count value become zero. By this technique only an

ஆய்வுச்சுருக்கம்

தகவல் அனுப்பப்பயன்படும் கட்டமைப்புகளையும் பாதைகளையும் கட்டுப்பாட்டில் வைப்பது, மூன்றாவரிடம் இருந்து தகவல்களையும், கட்டமைப்பையும் பாதுகாக்க முடியும்.

இந்த ஆராய்ச்சியில் முக்கியமாக டிடிஎல் என்ற ஒரு தகவல் தொகுப்பின் ஒரு பகுதியின் வாழ்நாள் மதிப்பை அடிப்படியாகக் கொண்டு அனுப்புனர்க்கும் பெருநர்க்கும் இடையிலான பாதையை மற்றும் கட்டமைப்புகளை தேர்வு செய்யப்படுகிறது.

மேலும் பிறரிடம் இருந்து தகவல்களை மேலும் பாதுகாக்க டிஇஎஸ் என்கின்ற தகவல் உருமாற்றும் தந்திரம் உபயோகப்படுத்தி தகவல்கள் உருமாற்றி பாதைகளையும், கட்டமைப்புகளையும் தேர்வு செய்யும் திரனானது. பெரும்பாலான தகவல் திருடல் மற்றும் தடுத்தல்களில் இருந்து தகவல்களை காக்கின்றது.

மேர்கூரியவற்றால் தகவல் பறிமாற்றமான மிகவும் நம்பகத் தன்மையாக உள்ளது அனுப்புனர் மற்றும் பெறுநர் மட்டுமே தகவல்களை அறியும்ப்படியான பாதுகாப்பாக அனுப்ப ஏதுவாக உள்ளது.

ACKNOWLEDGEMENT

I express my profound gratitude to our Chairman **Padmabhusan Arutselver Dr. N. Mahalingam B.Sc, F.I.E** for giving this great opportunity to pursue this course.

I would like to begin by thanking **to Dr. S,Ramachandran, Ph.D** *Principal* for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Dr. S.Thangasamy, Ph.D** Dean , Department of Computer Science and Engineering, for his precious suggestions.

I extend my sincere thanks to **Ms. V.Vanitha, M.E., Project Coordinator,** Department of computer Science and Engineering for her valuable suggestions and guidance.

I express my deep sense of gratitude and gratefulness to my guide **Mr. K.Sivan Arul Selvan, M.E,** Department of computer Science and Engineering for his supervision, tremendous patience, active involvement and guidance.

I would like to convey my honest thanks to all **Teaching** staff and **Non Teaching** staff of the department for their support.

I dedicate this project work to my **parents** for no reasons but feeling from bottom of my heart, without their love this work wouldn't possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

TTL	Time To Live
DES	Data Encryption Standard
FI	Forwarding Infrastructure

LIST OF SYMBOLS

LS	**left shift**
RS	**right shift**
k	**key value**
h	**hash function**
S	**substitution box**
H	**host**
E,D	**encryption and decryption**
PC	**permuted choice**

CHAPTER 1

INTRODUCTION TO NETWORK SECURITY

1.1 OVERVIEW OF NETWORK SECURITY

Network security consists of the provisions made in an underlying Computer Network infrastructure, policies adopted by the network administrator to protect the network and the network accessible resources from unauthorized access and the effectiveness (or lack) of these measures combined together. Securing network infrastructure is like securing possible entry points of attacks on a country by deploying appropriate defense. Computer security is more like providing means to protect a single PC against outside intrusion. The former is better and practical to protect the civilians from getting exposed to the attacks. The preventive measures attempt to secure the access to individual computers. The network itself thereby protecting the computers and other shared resources such as printers, network-attached storage connected by the network. Attacks could be stopped at their entry points before they spread. As opposed to this, in computer security the measures taken are focused on securing individual computer hosts. A computer host whose security is compromised is likely to infect other hosts connected to a potentially unsecured network. A computer host's security is vulnerable to users with higher access privileges to those hosts.

Network security starts from authenticating any user, most likely a username and a password. Once authenticated, a stateful firewall enforces access policies such as what services are allowed to be accessed by the network users. Though effective to prevent unauthorized access, this component fails to check potentially harmful contents such as computer worms being transmitted over the network. An intrusion prevention system (IPS) helps detect and prevent such malware. IPS also monitors for suspicious network traffic for contents, volume and anomalies to protect the network from attacks such as denial of service. Communication between two hosts using the network could be encrypted to maintain privacy. Individual events occurring on the network could be tracked for audit purposes and for a later high level analysis.

Until modern times cryptography referred almost exclusively to encryption, which is the process of converting ordinary information (plaintext) into unintelligible text (ciphertext).Decryption is the reverse process that is moving from the unintelligible ciphertext back to plaintext. A cipher is a pair of algorithms which create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a key. This is a secret parameter (ideally known only to the communicants) for a specific message exchange context. Keys are important, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore less than useful for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.

Some use the terms *cryptography* and *cryptology* .*cryptography* to refer specifically to the use and practice of cryptographic techniques and *cryptology* to refer to the combined study of cryptography and cryptanalysis. Before cryptography was concerned solely with message confidentiality (i.e., encryption) — conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge (namely the key needed for decryption of that message). Encryption was used to ensure secrecy in communications .In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs and secure computation, among others. The main classical cipher types are transposition ciphers, which rearrange the order of letters in a message (e.g., 'hello world' becomes 'ehlol owrdl' in a trivially simple rearrangement scheme), and substitution ciphers ,which systematically replace letters or groups of letters with other letters or groups of letters (e.g., 'fly at once' becomes 'gmz bu podf' by replacing each letter with the one following it in the Latin alphabet).

The modern field of cryptography can be divided into symmetric key crptography and asymmetric key cryptography.

1.1.1 Symmetric-key cryptography

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key .The modern study of symmetric-key ciphers relates mainly to the study of block ciphers and stream ciphers and to their applications. A block cipher is, in a sense, a modern embodiment of Alberti's polyalphabetic cipher: block ciphers take as input a block of plaintext and a key, and output a block of ciphertext of the same size. Since messages are almost always longer than a single block, some method of knitting together successive blocks is required. Several have been developed, some with better security in one aspect or another than others. They are the modes of operation and must be carefully considered when using a block cipher in a cryptosystem.

The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs which have been designated cryptography standards by the US government (though DES's designation was finally withdrawn after the AES was adopted). Despite its deprecation as an official standard, DES (especially its still-approved and much more secure triple-DES variant) remains quite popular; it is used across a wide range of applications, from ATM encryption to e-mail privacy and secure remote access. Many other block ciphers have been designed and released, with considerable variation in quality Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state which changes as the cipher operates. That internal state is initially set up using the secret key material. RC4 is a widely used stream cipher; see Category: Stream ciphers· Block ciphers can be used as stream ciphers.

Cryptographic hash functions are a third type of cryptographic algorithm. They take a message of any length as input, and output a short, fixed length hash which can be that produce the same hash. MD4 is a long-used hash function which is now broken; MD5, a strengthened variant of MD4, is also widely used but broken in practice

1.1.2 Public-key cryptography

Symmetric-key cryptosystems use the same key for encryption and decryption of a message, though a message or group of messages may have a different key than others. A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all straight and secret. The difficulty of securely establishing a secret key between two communicating parties, when a secure channel doesn't already exist between them.

public-key (also, more generally, called *asymmetric key*) cryptography in which two different but mathematically related keys are used — a *public* key and a *private* key. A public key system is so constructed that calculation of one key (the 'private key') is computationally infeasible from the other (the 'public key'), even though they are necessarily related. Instead, both keys are generated secretly, as an interrelated pair.In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret. The *public key* is typically used for encryption, while the *private* or *secret key* is used for decryption. RSA, another public-key system.

1.2 Benefits of Network Security:

The various benefits of using Network Security are listed below

- ❖ Network Security Auditing.
- ❖ Systems Security Consultations.
- ❖ Penetration Testing Services.
- ❖ Customized Network Security Systems Design.

- Fast and Efficient Systems Installation.
- Professional Network Security Management and Support.
- Enhanced System Security For Sensitive Data.

CHAPTER 2

LITERATURE SURVEY

An *external attacker* does not control any compromised FI node but misuses the flexibility given by the FI. An external attacker can perform only the operations that a legitimate host can: insert a forwarding entry and send a packet. An *internal attacker* is an adversary who controls some compromised FI nodes. Ideally, we want to ensure that an external attacker cannot eavesdrop or impersonate a host or misuse an FI network to amplify the magnitude of a flooding attack.3 In the case of an internal attack, we want to ensure that an attacker who compromises an FI node cannot affect other traffic that is not forwarded through that compromised FI node.

In this existing system they transfer message for securing purpose they use encryption and decryption process. They use public key value so intermediate user also able to view message. Here two attacks will be occurring. That is Internal Attack and External attack.

2.1 EXISTING METHODS

2.1.1 Constrained ID Technique

Constrained IDs is our core technique, which prevents eaves-dropping, impersonation, and the construction of topologies that are not trees. Consider an FI node that updates the packet ID from id to id^1 We enforce a constraint on the structure of IDs such that the choice of or id^1 vice-versa.To implement the constraints, we divide id.key into two sub-fields: a constrained part (id.key.c)and an unconstrained part(id.key.u). When a packet is matched at an FI node, the con-strained part must match.

2.1.1.1 Constrained **IDs Rule**: A packet ID ,id,can be updated to id^1, if and only if either id^1.key.c=h_r(id.node,id.key.c) or id.key.c=h_l(id^1.node,id.key.c)hold. Functions h and h are cryptographic hash functions mapping N -bit strings to n-bit strings, where N is thesize of an ID excluding the un-constrained part of the key, and n is the size of the constrained part of the key. The properties we require of the cryptographichash functions: 1) strong collision resistance and 2)computational infeasibility of finding short cycles. Secure one-way hash functions, such as SHA providethese two properties. If it is clear from the context, we use id=h(id) and id=h(id) as a shorthand for id.key.c=h(id.node,id.key.c) and id.key.c=h(id.node,id.key.c),respectively.

Intuitively, a cryptographic hash function makes it hard for an adversary to construct malicious topologies such as loops.Since h and h are publicly-known hash functions, any FI nodeor host can check and enforce the constraints. If packet's ID, id is updated to id^1 and id^1=h(id), and that packet ID isright-constrained ; otherwise, we say that it is left_contrained.

In short, l-constrained entries are used to protect against eavesdropping and impersonation, whereas r-constrained entries are used to construct flexible topologies that are resistant to amplification attacks. Next, Note however that since we allow flexibility of choosing id.node, one can still construct confluences on end-hosts and FI nodes.

The rule that we use to constrain IDs results directly fromthe dual goal of achieving the desirable security properties andat the same time preserving the FI functionality.To enumerate several alternatives to constrain IDs considered.

1) Constraining the entire ID id^1 using id^1 .node would depend on id. Thiswould limit the flexibility of an end-user or third-party inchoosing the nodes along a path.
2) Constraining the entire id.key using some part of id' would be restrictive, as some FIs require control on the value of id.key.
3) Constraining id'.key using only id.key would allow an attacker to create confluences on FI nodes by mapping all the leaf IDs to the victim node.

2.1.2 Challenge-Response

challenge-response protocol is password authentication, where the challenge is asking for the password and the valid response is the correct password.Clearly an adversary that can eavesdrop on a password authentication can then authenticate itself in the same way. One solution is to issue multiple passwords, each of them marked with an identifier. The verifier can pick any of the identifiers, and the prover must have the correct password for that identifier. Assuming that the passwords are chosen independently, an adversary who intercepts one challenge-response message pair has no more chance of responding correctly to a different challenge than an adversary who has intercepted nothing.

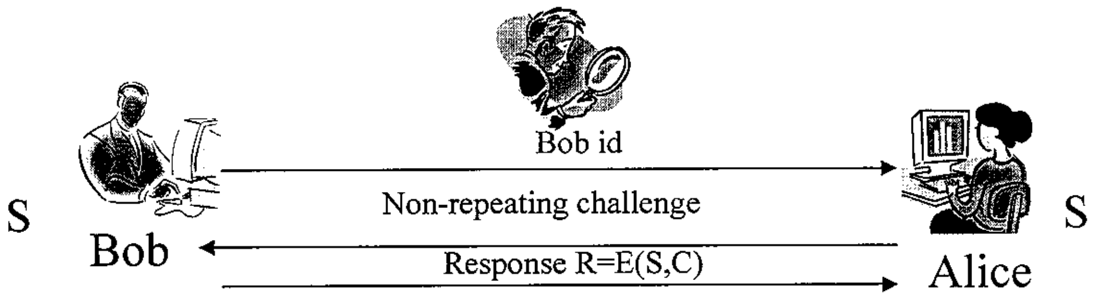


Fig 2.1: Challenge response technique

To ensure that an attacker cannot insert entries pointing to other benign end-hosts use the well-known challenge-response technique. FI nodes challenge the insertion of every forwarding entry using a simple three-way handshake.

The challenge-response protocol helps preventing amplification attacks on end-hosts since an attacker cannot insert an entry pointing to an arbitrary end-host it does not control. Hence, to replicate its traffic and direct it towards a particular host, the attacker must itself create a malicious ID-level topology, and link all leaves with an existing entry.

2.2 Existing System:

- An attacker should be able to eavesdrop on the traffic to an arbitrary host.
- An attacker should be able to amplify its attack on end-hosts using the FI.
- An attacker can any cause a small bounded attack on the FI.
- An attacker that has compromised an FI node can any affect traffic that the compromised FI node forwards.
- In this existing system they didn't avoid the hacks.

2.2 .1 Drawbacks:

In this existing system they transfer message for securing purpose they use encryption and decryption process. They use public key value so intermediate user also able to view message. Here two attacks will be occurring. That is Internal Attack and External attack.

2.3 Proposed System:

Improve the security that flexible communication infrastructures that provide a diverse set of operations (such as packet replication) allow. Our main goal in this paper is to show that FIs are no more vulnerable than traditional communication networks (such as IP networks) that do not export control on forwarding. To this end, we present several mechanisms that make these FIs achieve certain specific security properties; yet retain the essential features and efficiency of their original design. Our main defense technique, which is based on lightweight cryptographic constraints on forwarding entries, prevents several attacks including eavesdropping, loops, and traffic amplification.

2.3.1 Advantages of the Proposed System:

- Here they avoid attacks.
- Here they use public key value but intermediate can't able to view the message because final hop identify means after check the cost value.
- The cost value match means after only the user can able to view the message otherwise display the message Access denied.
- Here they use DES algorithm for encryption and decryption process.

CHAPTER 3
PROBLEM DEFINITION

Infrastructure have many types is there. That is DataRouter, Network Pointer, i3 and Forwarding Infrastructure. In this paper they analyzing about Forwarding Infrastructure. Here two attacks are occurring. That is External attack and internal attack. In this paper we use cryptographic constrain and find out the one constrain. In that constrain we apply all type of Infrastructure then its applicable.

In this existing system they transfer message for securing purpose they use encryption and decryption process. They use public key value so intermediate user also able to view message. Here two attacks will be occurring. That is Internal Attack and External attack.

An *external attacker* does not control any compromised FI node but misuses the flexibility given by the FI. An external attacker can perform only the operations that a legitimate host can: insert a forwarding entry and send a packet. An *internal attacker* is an adversary who controls some compromised FI nodes. Ideally, we want to ensure that an external attacker cannot eavesdrop or impersonate a host or misuse an FI network to amplify the magnitude of a flooding attack.3 In the case of an internal attack, we want to ensure that an attacker who compromises an FI node cannot affect other traffic that is not forwarded through that compromised FI node.

CHAPTER 4
PROPOSED METHODOLOGY

Infrastructure have many types is there. That is Data Router, Network Pointer, i3 and Forwarding Infrastructure. In this paper they analyzing about Forwarding Infrastructure. Here two attacks are occurring. That is External attack and internal attack. In this paper we use cryptographic constrain and find out the one constrain. In that constrain we apply all type of Infrastructure then its applicable. Cryptographic constrain means Encryption and Decryption process. Here they use DES algorithm for Encryption and Decryption process. In this algorithm they use public key value. Here they avoid the existing drawbacks.

An *external attacker* does not control any compromised FI node but misuses the flexibility given by the FI. An external attacker can perform only the operations that a legitimate host can: insert a forwarding entry and send a packet. An *internal attacker* is an adversary who controls some compromised FI nodes. Ideally, we want to ensure that an external attacker cannot eavesdrop or impersonate a host or misuse an FI network to amplify the magnitude of a flooding attack.3 In the case of an internal attack, we want to ensure that an attacker who compromises an FI node cannot affect other traffic that is not forwarded through that compromised FI node.

4.1 Threat Model

We consider two attacker types: internal and external attackers. An *external attacker* does not control any compromised FI node but misuses the flexibility given by the FI. An external attacker can perform only the operations that a legitimate host can: insert a forwarding entry and send a packet. An *internal attacker* is an adversary who controls some compromised FI nodes.

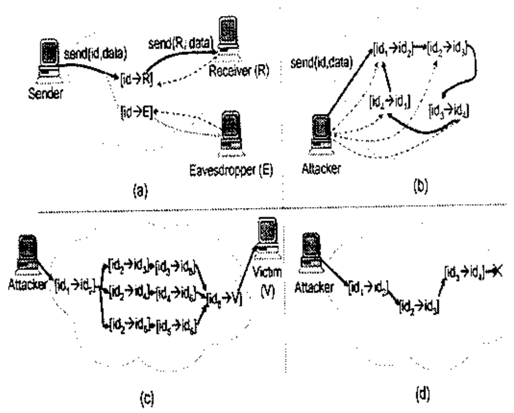


Fig 4.1: Attack examples. (a) Eavesdropping. (b) Cycle. (c) End-host confluence. (d) Dead end.

Cycles involving FI nodes. An attacker can form a loop by inserting forwarding entries [see Fig. 1(b)]. This cycle indefinitely consume FI resources.

Dead-ends. An attacker can construct a chain of forwarding entries, or even a multicast tree, which do not point to a valid end-host [see Fig. 1(d)]. Data packets sent on such a topology would be forwarded and replicated only to be dropped at the dead ends.

FI confluence. An attacker can refine a dead-ends attack by constructing a multicast tree with *m* leaves, all pointing to a victim FI node. For every packet sent by the attacker, the victim will receive *m* duplicates.

Ideally, we want to ensure that an external attacker cannot eavesdrop or impersonate a host or misuse an FI net- work to amplify the magnitude of a flooding attack. In the case of an internal attack, we want to ensure that an attacker who compromises an FI node cannot affect other traffic that is not forwarded through that compromised FI node.

4.3 PROPOSED SYSTEM STRUCTURE

Here we consider the source routing that means the user can specify the path from source to destination through the intermediate nodes and also use public key value but intermediate can't able to view the message because final hop identify means after check the cost value. The cost value match means after only the user can able to view the message otherwise display the message Access denied. Then use DES algorithm for encryption and decryption process.

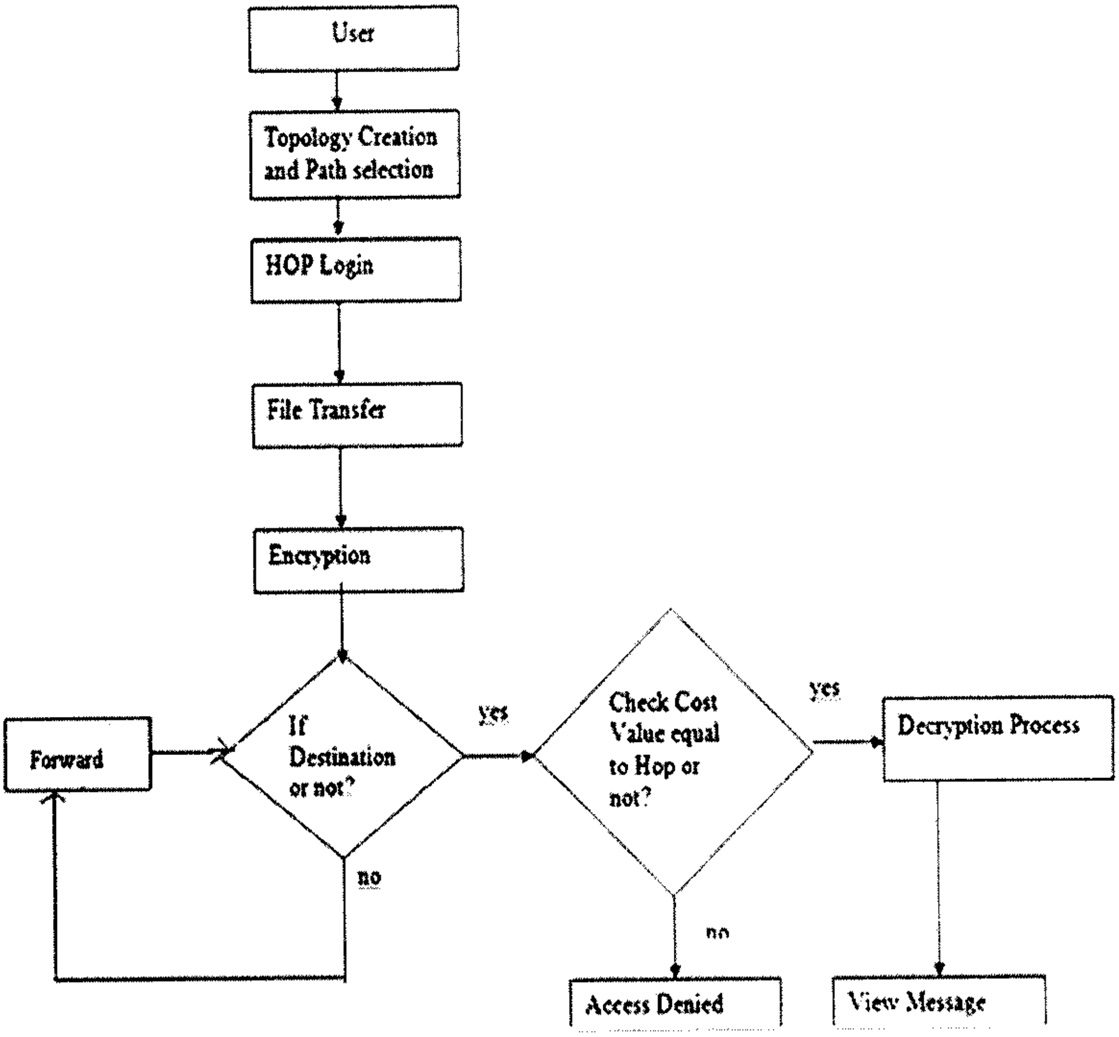


Fig 4.3: Proposed system structure

4.3.1 Constructing a acyclic Topology and Path Selection

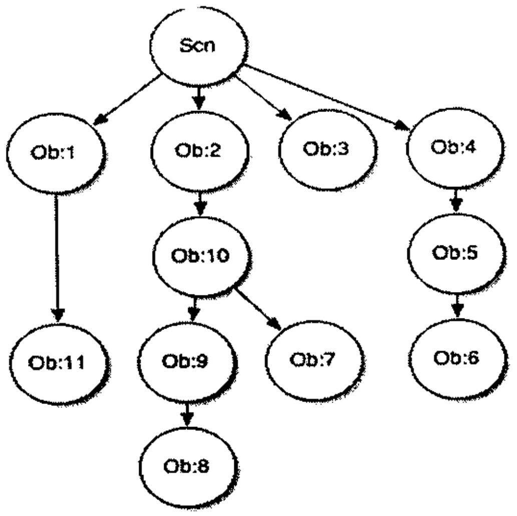


Fig 4.4 acyclic topology

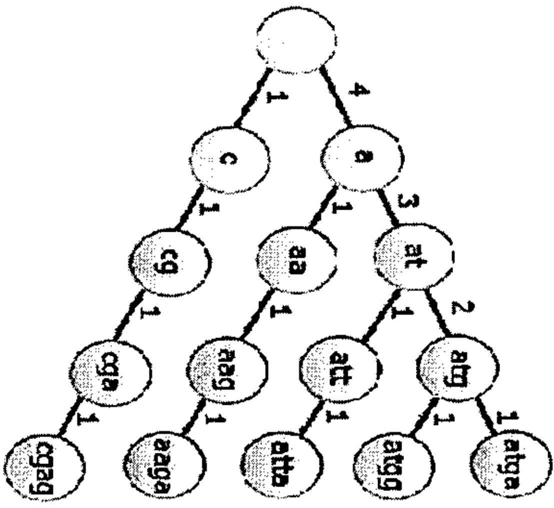


Fig 4.5 path selection using TTL value

Firstly construct a acyclic topology. This topology is constructed by getting the names of the nodes and the connections among the nodes as input from the user. While getting each of the nodes, their associated port and ip address is also obtained. For successive nodes, the node to which it should be connected to it parent node. While adding nodes, comparison will be done so that there would be no node duplication. Then identify the source and the destinations. In this module, I find the possible path for each of the destinations from the source. After finding the possible paths, find the cost associated with each of those paths.

4.3.2 Hop Login:

In this module, we Login the Hops for Message Transfer and validate hop name, hop password and port number. These value are equal means separate server listen for transfer Message.

4.3.3 Defense Mechanism

In this module a file to be encrypted is chosen. The chosen file is given as the message transfer to Destination Hop. A key governs the encryption and the key must be of 8 digits. The Next Hop with in the Network means entered the correct key value after the header information decrypted to view the user. The users can rights to modify the path value. After again encrypted the header information then transfer to next Hop. These it continues up to final Hop identify. In this module decrypt to view header information. The next Hop users enter the 8 digit key value. The key value same means only the header information decrypt to view the user. After the user change the path value.

4.3.4 Message View

After changing path value identified that is final Hop means check the cost value.The cost value is same means message decrypt to view the destination. Not same cost value means its display Access Denied.

CHAPTER 5

IMPLEMENTATION

This project is the implementation of DES algorithm by which a file can be encrypted and decrypted. The encryption and decryption can be done under provision of a key. The key must be a 8 digit integer. DES encrypts and decrypts data in 64-bit blocks, using a 64-bit key (although the effective key strength is only 56 bits, as explained below). It takes a 64-bit block of plaintext as input and outputs a 64-bit block of cipher text. Since it always operates on blocks of equal size and it uses both permutations and substitutions in the algorithm, DES is both a block cipher and a product cipher.

DES has 16 rounds, meaning the main algorithm is repeated 16 times to produce the cipher text. It has been found that the number of rounds is exponentially proportional to the amount of time required to find a key using a brute-force attack. So as the number of rounds increases, the security of the algorithm increases exponentially.

5.1 FORWARDING INFRASTRUCTURE MODEL

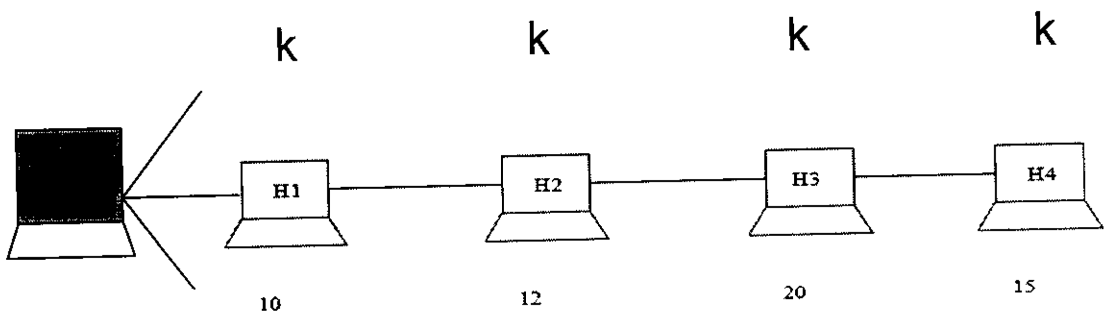


Fig 5.1: Architecture of forwarding infrastructure

TTL value from H1 to H4 = 10+12+20+15 =57

Updated TTL value at H1=57-10

Updated TTL value at H2=47-12

=35

Updated TTL value at H3=35-20

=15

Updated TTL value at H4=15-15

=0

5.1.1 Create Topology and Path Selection

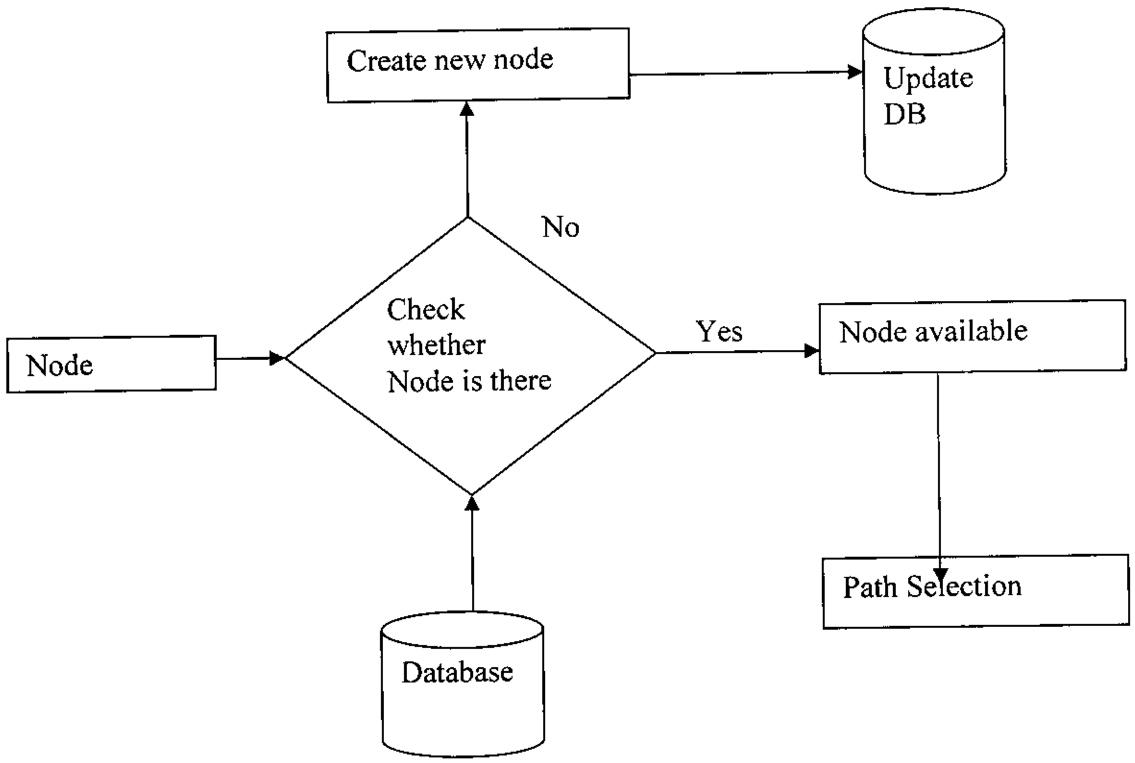


Fig 5.2: Flowchart for topology creation and path selection

In the Forwarding infrastructure model, consider 4 hosts. They are H1, H2, H3, and H4. These are constructed by getting the name of nodes and connections among the nodes as input from the user. And these values are stored in the database. After that the source routing is applies that means user can specify the path from source to destination.

Here all the users use the same key value. but unauthorized user can't view the message because messages are decrypted when hop count becomes zero.

5.2 Hop Login

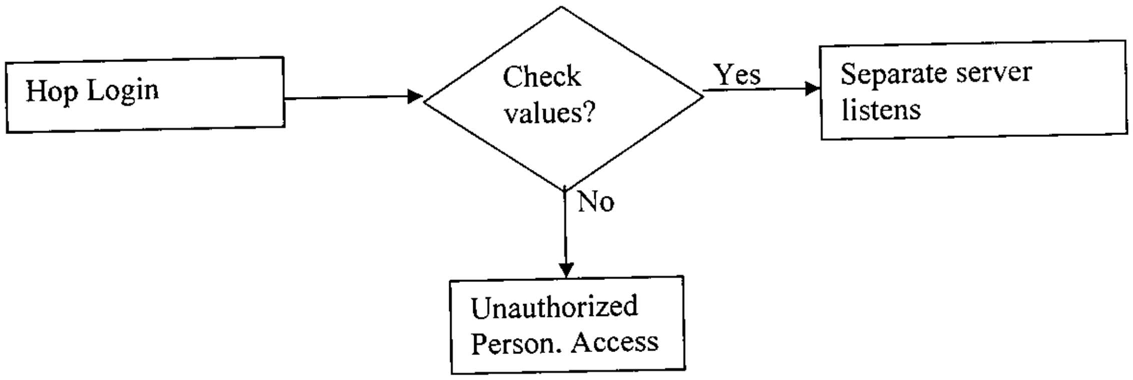


Fig 5.3: login the hops for message transfer

In the forwarding infrastructure model all the 4 hops are login for message transfer. If the user specify the source is H1 and destination is H4, all intermediate nodes like H2 and H3 are login including H1 and H4. Check the entering values like IP address hop name and port number with stored values in database .If they are equals separate servers are listen for transferring the message up to the destination.

5.3 Encryption and Decryption

Nowadays when more and more sensitive information is stored on computers and transmitted over the Internet, we need to ensure information security and safely.

One of the most common uses of encryption is encrypting emails. Sending sensitive messages, documents and files over the Internet is like sending a postcard as all emails are transmitted in an unsecured form. It doesn't depend on if you send emails via public and private networks. Your message is totally open to interception by anyone along the way .Even if you connect to your server and send your emails via SSL, it only means that email reaches your server, your email service provider can see it. Then your server usually sends your email to the recipient in an unsecured way and your email can also be easily seen by anyone.

Private network, where email goes directly to a mail server and resides there until it is retrieved, also doesn't provide necessary security level, as you email can be seen Of course, you may believe that your personal email does not contain any private information, but everyone has got something to keep in secret from his family, neighbors or colleagues. It could be financial, sexual, social, political, or professional secrets. There is really only one sure way to protect Your email privacy-using encryption.

Sending sensitive messages and files over the Internet is very dangerous as all emails are transmitted in an unsecured form. If you need to send sensitive information Over the internet you should encrypt it first. Encryption and Decryption Pro allows you easily encrypt and decrypt your messages and files. For more security you can use multiple encryption. If you want to send sensitive information via email, simply paste the encrypted text into your email or attach the encrypted file - the entire recipient has to do is to decrypt your text or file.

The DES (Data Encryption Standard) algorithm is the most widely used encryption algorithm in the world. For many years, and among many people, "secret code making" and DES have been synonymous. And despite the recent coup by the Electronic Frontier Foundation in creating a $220,000 machine to crack DES-encrypted messages, DES will live on in government and banking for years to come through a life- extending version called "triple-DES."

DES is the block cipher algorithm— an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another cipher text bit string of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits, and it is usually quoted as such.

Three phases are:

- Initial permutation
- Round function
- Inverse initial permutation

There are 16 identical stages of processing, termed *rounds*. There is also an initial and final permutation, termed *IP* and *FP*, which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have almost no cryptographic significance, but were apparently included in order to facilitate loading blocks in and out of mid-1970s hardware, as well as to make DES run slower in software.

Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes — the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.

The *F-function* scrambles half a block together with some of the key. The output from the F-function is then combined with the other half of the block, and the halves are swapped before the next round. After the final round, the halves are not swapped; this is a feature of the Feistel structure which makes encryption and decryption similar processes

Round function:

- Expansion : the 32-bit half-block is expanded to 48 bits.

- Key mixing : the result is combined with a subkey using an XOR operation. Sixteen 48-bit subkeys — one for each round — are derived from the main key

- Substitution: after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by s-boxes.
- permutation :finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation.

DES is a ***block cipher***--meaning it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Thus DES results in a ***permutation*** among the 2^64 (read this as: "2 to the 64th power") possible arrangements of 64 bits, each of which may be either 0 or 1. Each block of 64 bits is divided into two blocks of 32 bits each, a left half block **L** and a right half **R**. (This division is only used in certain operations.)

5.3.1 DES algorithm

1 process the key.

1.1 Get a 64-bit key from the user. (Every 8th bit (the least significant bit of each byte) is considered a parity bit. For a key to have correct parity, each byte should contain an odd number of "1" bits.)

1.2 Calculate the key shedule.

1.3 Perform the following permutation on the 64-bit key. (The parity bits are discarded reducing the key to 56 bits.

Permuted Choice 1 (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

1.2.2 Split the permuted key into two halves. The first 28 bits are called C[0] and the last 28 bits are called D[0].

1.2.3 Calculate the 16 sub keys. Start with i = 1.

1.2.3.1 Perform one or two circular left shifts on both C[i-1] and D[i-1] to get C[i] and D[i], respectively. The number of shifts per iteration are given in the table below.

1.2.3.2 Permute the concatenation C[i]D[i] as indicated below. This will yield K[i], which is 48 bits long.

Permuted Choice 2 (PC-2)

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

1.2.3.3 Loop back to 1.2.3.1 until K[16] has been calculated.

2 Process a 64-bit data block.

2.1 Get a 64-bit data block. If the block is shorter than 64 bits, it should be padded as appropriate for the application.

2.2 Perform the following permutation on the data block.

Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6

64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

2.3 Split the block into two halves. The first 32 bits are called L[0], and the last 32 bits are called R[0].

2.4 Apply the 16 sub keys to the data block. Start with i = 1.

2.4.1 Expand the 32-bit R[i-1] into 48 bits according to the bit-selection function below.

Expansion (E)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

2.4.2 Exclusive-or E(R[i-1]) with K[i].

2.4.3 Break E(R[i-1]) xor K[i] into eight 6-bit blocks. Bits 1-6 are B[1], bits 7-12 are B[2], and so on with bits 43-48 being B[8].

2.4.4 Substitute the values found in the S-boxes for all B[j]. Start with j = 1. All values in the S-boxes should be considered 4 bits wide.

2.4.4.1 Take the 1st and 6th bits of B[j] together as a 2-bit value (call it m) indicating the row in S[j] to look in for the substitution.

2.4.4.2 Take the 2nd through 5th bits of B[j] together as a 4-bit value (call it n) indicating the column in S[j] to find the substitution.

2.4.4.3 Replace B[j] with S[j][m][n].

Substitution Box 1 (S[1])

14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13

S[2]

15 1 8 14 6 11 3 4 9 7 2 13 12 0 5 10
3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9

S[3]

10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12

S[4]

7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14

S[5]

2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3

S[6]

12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13

S[7]

4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12

S[8]

13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11

2.4.4.4 Loop back to 2.4.4.1 until all 8 blocks have been replaced.

2.4.5 Permute the concatenation of B[1] through B[8] as indicated below.

Permutation P

16 7 20 21
29 12 28 17
1 15 23 26
5 18 31 10
2 8 24 14
32 27 3 9
19 13 30 6
22 11 4 25

2.4.6 Exclusive-or the resulting value with L[i-1]. Thus, all together, your R[i] = L[i-1] xor P(S[1](B[1])...S[8](B[8])), where B[j] is a 6-bit block of E(R[i-1]) xor K[i]. (The function for R[i] is more concisely written as, R[i] = L[i-1] xor f(R[i-1], K[i]).)

2.4.7 L[i] = R[i-1].

2.4.8 Loop back to 2.4.1 until K[16] has been applied.

2.5 Perform the following permutation on the block R[16]L[16].

Final Permutation (IP**-1)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

```
Key schedule:
C[0]D[0] = PC1(key)
for 1 <= i <= 16
C[i] = LS[i](C[i-1])
D[i] = LS[i](D[i-1])
K[i] = PC2(C[i]D[i])

Encipherment:
L[0]R[0] = IP(plain block)
for 1 <= i <= 16
L[i] = R[i-1]
R[i] = L[i-1] xor f(R[i-1], K[i])
cipher block = FP(R[16]L[16])

Decipherment:
R[16]L[16] = IP(cipher block)
for 1 <= i <= 16
R[i-1] = L[i]
L[i-1] = R[i] xor f(L[i], K[i])
plain block = FP(L[0]R[0])
```

Example: Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
L = 0000 0001 0010 0011 0100 0101 0110 0111
R = 1000 1001 1010 1011 1100 1101 1110 1111

The first bit of **M** is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But, as you will see, the eight bits just mentioned get eliminated when we create subkeys.

Example: Let **K** be the hexadecimal key **K** = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

The DES algorithm uses the following steps:

Step 1: Create 16 subkeys, each of which is 48-bits long.

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K+**. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18

19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Example: From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

we get the 56-bit permutation

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Next, split this key into left and right halves, $\boldsymbol{C_0}$ and $\boldsymbol{D_0}$, where each half has 28 bits.

Example: From the permuted key **K+**, we get

$\boldsymbol{C_0}$ = 1111000 0110011 0010101 0101111
$\boldsymbol{D_0}$ = 0101010 1011001 1001111 0001111

With $\boldsymbol{C_0}$ and $\boldsymbol{D_0}$ defined, we now create sixteen blocks $\boldsymbol{C_n}$ and $\boldsymbol{D_n}$, $1<=n<=16$. Each pair of blocks $\boldsymbol{C_n}$ and $\boldsymbol{D_n}$ is formed from the previous pair $\boldsymbol{C_{n-1}}$ and $\boldsymbol{D_{n-1}}$, respectively, for n = 1, 2, ..., 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2

15	2
16	1

This means, for example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Example: From original pair pair C_0 and D_0 we obtain:

C_0 = 1111000011001100101010101111
D_0 = 0101010101100110011110001111

C_1 = 1110000110011001010101011111
D_1 = 1010101011001100111100011110

C_2 = 1100001100110010101010111111
D_2 = 0101010110011001111000111101

C_3 = 0000110011001010101011111111
D_3 = 0101011001100111100011110101

C_4 = 0011001100101010101111111100
D_4 = 0101100110011110001111010101

C_5 = 1100110010101010111111110000
D_5 = 0110011001111000111101010101

C_6 = 0011001010101011111111000011
D_6 = 1001100111100011110101010101

C_7 = 1100101010101111111100001100
D_7 = 0110011110001111010101010110

C_8 = 0010101010111111110000110011
D_8 = 1001111000111101010101011001

C_9 = 0101010101111111100001100110
D_9 = 0011110001111010101010110011

C_{10} = 0101010111111110000110011001
D_{10} = 1111000111101010101011001100

C_{11} = 0101011111111000011001100101
D_{11} = 1100011110101010101100110011

C_{12} = 0101111111100001100110010101
D_{12} = 0001111010101010110011001111

C_{13} = 0111111110000110011001010101
D_{13} = 0111101010101011001100111100

C_{14} = 1111111000011001100101010101
D_{14} = 1110101010101100110011110001

C_{15} = 1111100001100110010101010111
D_{15} = 1010101010110011001111000111

C_{16} = 1111000011001100101010101111
D_{16} = 0101010101100110011110001111

We now form the keys K_n, for $1<=n<=16$, by applying the following permutation table to each of the concatenated pairs C_nD_n. Each pair has 56 bits, but **PC-2** only uses 48 of these.

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of C_nD_n.

Example: For the first key we have C_1D_1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

which, after we apply the permutation **PC-2**, becomes

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

For the other keys we have

K_2 = 011110 011010 111011 011001 110110 111100 100111 100101
K_3 = 010101 011111 110010 001010 010000 101100 111110 011001
K_4 = 011100 101010 110111 010110 110110 110011 010100 011101
K_5 = 011111 001110 110000 000111 111010 110101 001110 101000
K_6 = 011000 111010 010100 111110 010100 000111 101100 101111
K_7 = 111011 001000 010010 110111 111101 100001 100010 111100
K_8 = 111101 111000 101000 111010 110000 010011 101111 111011
K_9 = 111000 001101 101111 101011 111011 011110 011110 000001
K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111
K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110
K_{12} = 011101 010111 000111 110101 100101 000110 011111 101001
K_{13} = 100101 111100 010111 010001 111110 101011 101001 000001
K_{14} = 010111 110100 001110 110111 111100 101110 011100 111010
K_{15} = 101111 111001 000110 001101 001111 010011 111100 001010
K_{16} = 110010 110011 110110 001011 000011 100001 011111 110101

So much for the subkeys. Now we look at the message itself.

Step 2: Encode each 64-bit block of data.

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Example: Applying the initial permutation to the block of text **M**, given previously, we get

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.

Next divide the permuted block **IP** into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.

Example: From **IP**, we get L_0 and R_0

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

We now proceed through 16 iterations, for 1<=*n*<=16, using a function *f* which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2).** Then for **n** going from 1 to 16 we calculate

$L_n = R_{n-1}$
$R_n = L_{n-1} + f(R_{n-1}, K_n)$

This results in a final block, for n = 16, of $L_{16}R_{16}$. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation *f*.

Example: For n = 1, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
L_1 = R_0 = 1111 0000 1010 1010 1111 0000 1010 1010
$R_1 = L_0 + f(R_0, K_1)$

It remains to explain how the function *f* works. To calculate *f*, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1} . We'll call the use of this selection table the function **E**. Thus **E**(R_{n-1}) has a 32 bit input block, and a 48 bit output block.

Let **E** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of **E**(R_{n-1}) are the bits in positions 32, 1 and 2 of R_{n-1} while the last 2 bits of **E**(R_{n-1}) are the bits in positions 32 and 1.

Example: We calculate **E**(R_0) from R_0 as follows:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

E(R_0) = 011110 100001 010101 010101 011110 100001 010101 010101

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the *f* calculation, we XOR the output **E**(R_{n-1}) with the key K_n:

K_n + **E**(R_{n-1}).

Example: For K_1 , **E**(R_0), we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

E(R_0) = 011110 100001 010101 010101 011110 100001 010101 010101

011000 010001 011110 111010 100001 100110 010100 100111

We have not yet finished calculating the function f. To this point we have expanded R_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key K_n . We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different S box. Located at that address will be a 4 bit number. This 4 bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the S boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$\boldsymbol{K_n} + \mathrm{E}(\boldsymbol{R_{n-1}}) = \boldsymbol{B_1B_2B_3B_4B_5B_6B_7B_8}$,

where each $\boldsymbol{B_i}$ is a group of six bits. We now calculate

$\boldsymbol{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)}$

where $\boldsymbol{S_i(B_i)}$ referres to the output of the $\boldsymbol{i}$-th S box.

To repeat, each of the functions ***S1, S2,..., S8***, takes a 6-bit block as input and yields a 4-bit block as output. The table to determine $\boldsymbol{S_1}$ is shown and explained below:

S1

Row	Column Number															
No.	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**14**	**4**	**13**	**1**	**2**	**15**	**11**	**8**	**3**	**10**	**6**	**12**	**5**	**9**	**0**	**7**
1	**0**	**15**	**7**	**4**	**14**	**2**	**13**	**1**	**10**	**6**	**12**	**11**	**9**	**5**	**3**	**8**
2	**4**	**1**	**14**	**8**	**13**	**6**	**2**	**11**	**15**	**12**	**9**	**7**	**3**	**10**	**5**	**0**
3	**15**	**12**	**8**	**2**	**4**	**9**	**1**	**7**	**5**	**11**	**3**	**14**	**10**	**0**	**6**	**13**

If $\boldsymbol{S_1}$ is the function defined in this table and $\boldsymbol{B}$ is a block of 6 bits, then $\boldsymbol{S_1(B)}$ is determined as follows: The first and last bits of $\boldsymbol{B}$ represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be $\boldsymbol{i}$. The middle 4 bits of $\boldsymbol{B}$

represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be ***j***. Look up in the table the number in the ***i***-th row and ***j***-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output ***S_1(B)*** of ***S_1*** for the input ***B***. For example, for input block ***B*** = 011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence ***S_1***(011011) = 0101.

The tables defining the functions ***S_1,...,S_8*** are the following:

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Example: For the first round, we obtain as the output of the eight S boxes:

$\boldsymbol{K_1}$ + E($\boldsymbol{R_0}$) = 011000 010001 011110 111010 100001 100110 010100 100111.

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

The final stage in the calculation of f is to do a permutation **P** of the S-box output to obtain the final value of f:

f = **P**($S_1(B_1)S_2(B_2)...S_8(B_8)$)

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Example: From the output of the eight S boxes:

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

we get

f = 0010 0011 0100 1010 1010 1001 1011 1011

$R_1 = L_0 + f(R_0, K_1)$

```
=     1100    1100    0000    0000    1100    1100    1111    1111
+     0010    0011    0100    1010    1010    1001    1011    1011
= 1110 1111 0100 1010 0110 0101 0100 0100
```

In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds. At the end of the sixteenth round we have the blocks L_{16} and R_{16}. We then ***reverse*** the order of the two blocks into the 64-bit block

$R_{16}L_{16}$

and apply a final permutation **IP**$^{-1}$ as defined by the following table:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, on the 16th round,

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

We reverse the order of these two blocks and apply the final permutation to

$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

which in hexadecimal format is

85E813540F0AB405.

This is the encrypted form of **M** = 0123456789ABCDEF: namely, **C** = 85E813540F0AB405.

Decryption is simply the inverse of encryption, follwing the same steps as above, but reversing the order in which the subkeys are applied.

5.4: Message View

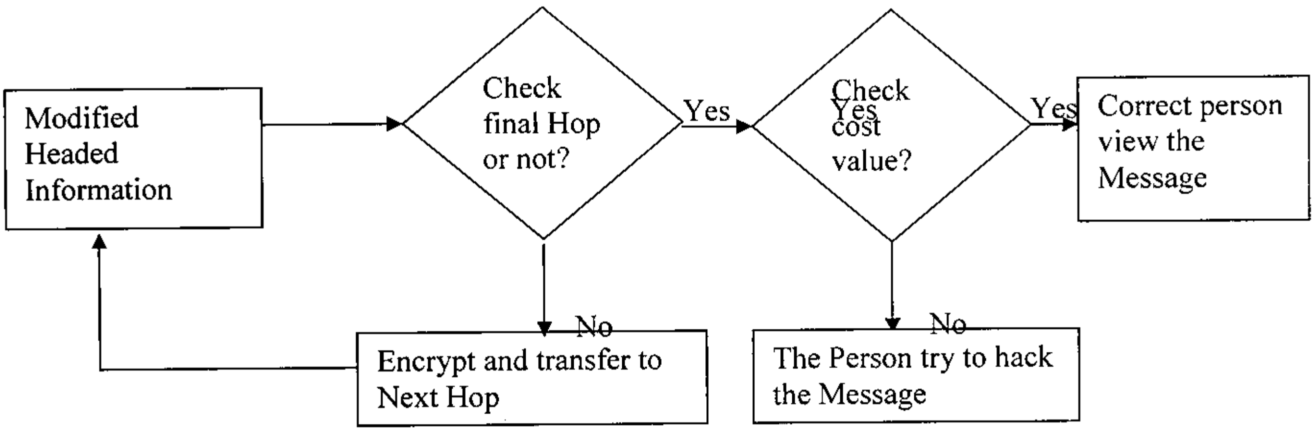

Modified Headed Information
Check final Hop or not?
Yes
Check cost value?
Yes
Yes
Correct person view the Message
No
Encrypt and transfer to Next Hop
No
The Person try to hack the Message

Fig 5.4 : Message view by the authorized person

If the updated value within in the packet header information compared with the entering cost value of each host at the time of topology creation. If they are equals means hop count becomes zero. so the destination only can view the message.

CHAPTER 6

6.1 CONCLUSION

Giving hosts control over forwarding in the infrastructure has become one of the promising approaches in designing flexible network architectures. In this paper, we addressed the security concerns of these forwarding infrastructures.

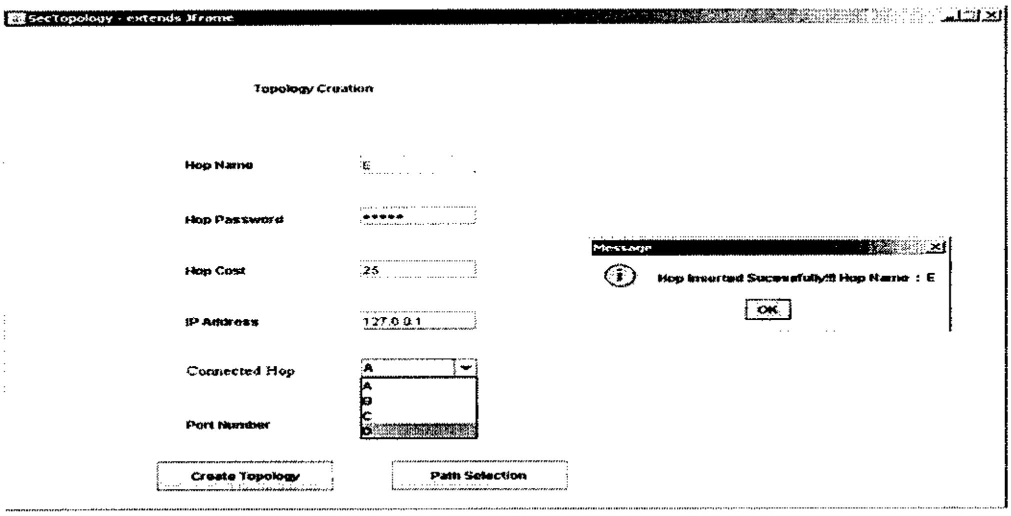
We presented a general FI model, analyzed potential security vulnerabilities, and presented a defense mechanism based on TTL value. Our key defense mechanism, based on TTL value, prevent external and internal attacks on end-host as well as intermediate nodes. By this system, if public key value is available the unauthorized person can't view the message

6.2 FUTURE WORK

In the process of designing security mechanisms for FIs, we have leveraged a defense mechanism based on TTL value. By this mechanism overcome the security problems on end-host as well as intermediate nodes. But here we consider only one path from source to destination. As a future work we wish to provide all possible paths and also select a best path for reaching to destination based upon the smallest TTL value.

APPENDIX A

SCREEN SHOTS


SecTopology - extends JFrame

Topology Creation

Hop Name: E

Hop Password: *****

Hop Cost: 25

IP Address: 127.0.0.1

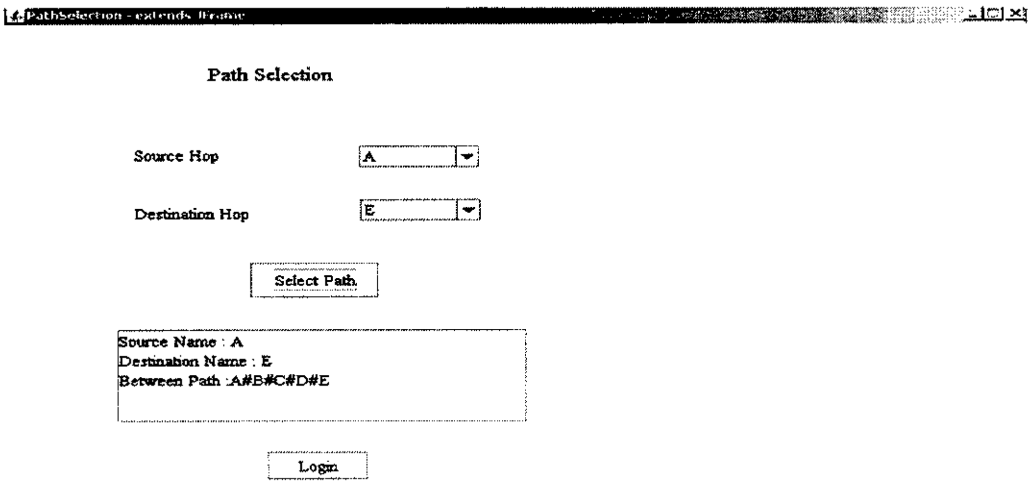
Connected Hop: A

Port Number

Message

Hop Inserted Successfully!! Hop Name : E

CREATE THE TOPOLOGY


PathSelection - extends JFrame

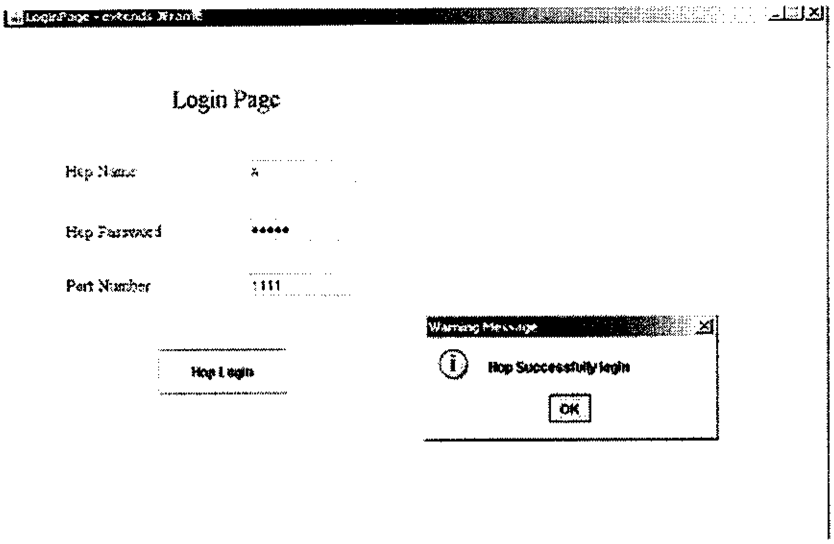
Path Selection

Source Hop: A

Destination Hop: E

```
Source Name : A
Destination Name : E
Between Path :A#B#C#D#E
```

SELECT THE PATH FROM SOURCE TO DESTINATION


LoginPage - extends JFrame

Login Page

Hop Name: a

Hop Password: *****

Port Number: 1111

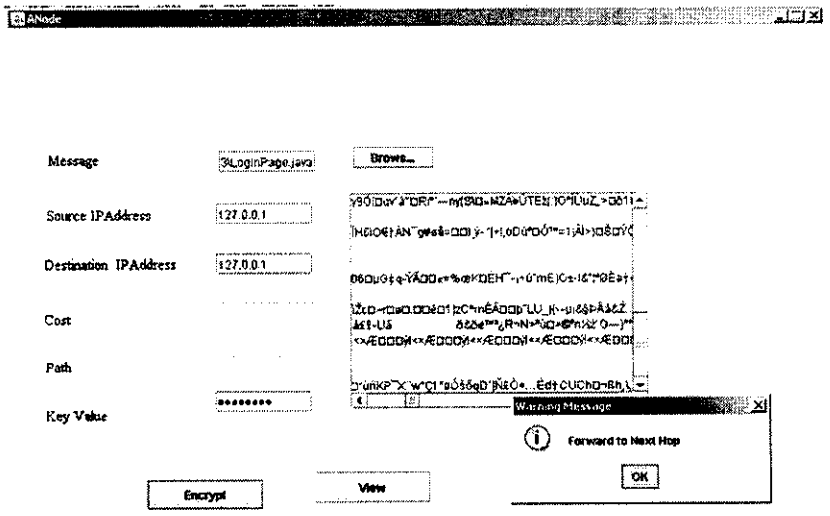
Hop Login

Warning Message

Hop Successfully login

OK

LOGIN THE HOP FOR MESSAGE TRANSFER


ANode

Message: 3\LoginPage.java

Browse...

Source IPAddress: 127.0.0.1

Destination IPAddress: 127.0.0.1

Cost

Path

Key Value: ********

[illegible]

Encrypt

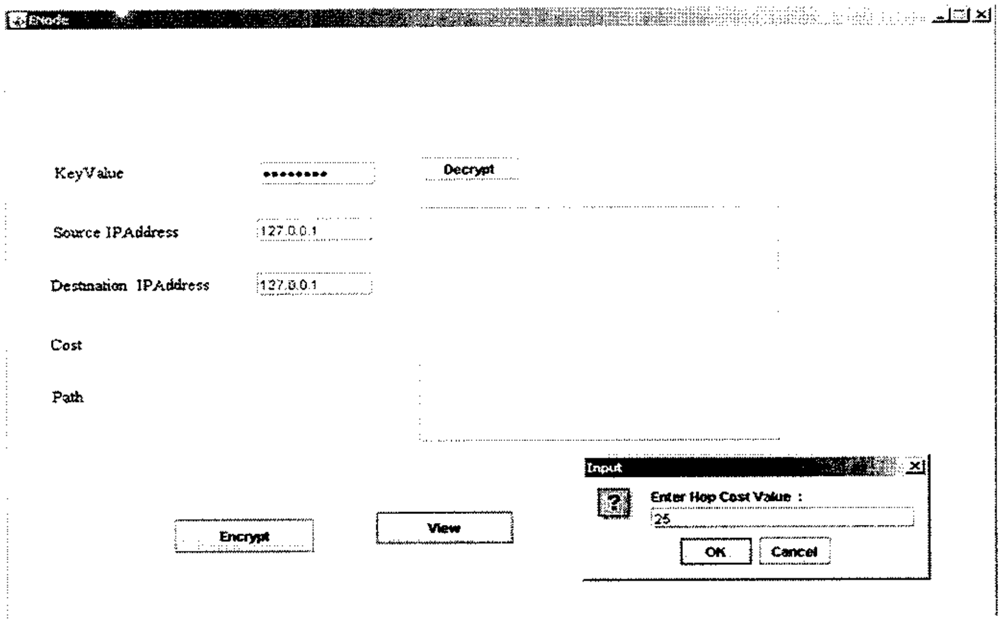
View

Warning Message

Forward to Next Hop

OK

MESSAGES ARE IN THE ENCRYPTED FORMAT



KeyValue ••••••••

Source IPAddress 127.0.0.1

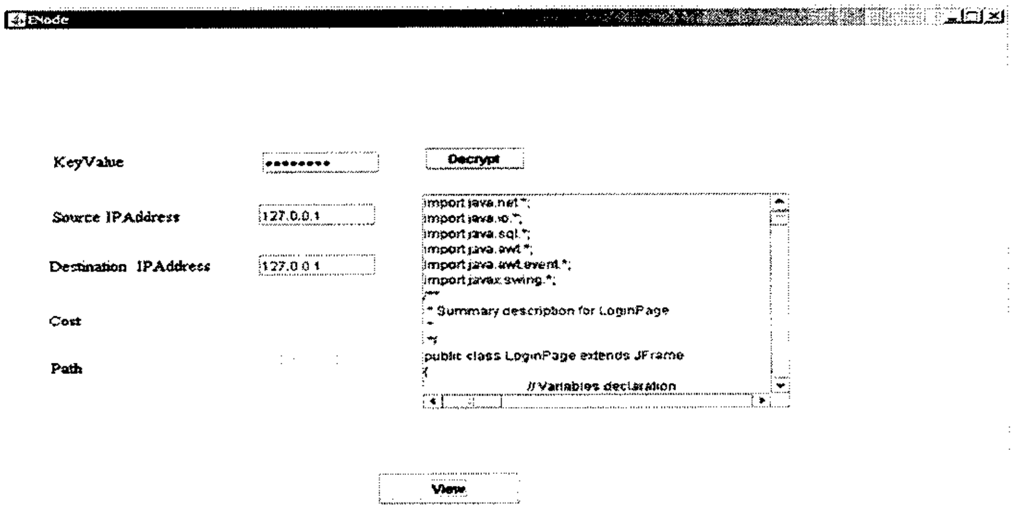
Destination IPAddress 127.0.0.1

Cost

Path

Enter Hop Cost Value : 25

FORWARD THE MESSAGE UPTO THE DESTINATION



KeyValue ••••••••

Source IPAddress 127.0.0.1

Destination IPAddress 127.0.0.1

Cost

Path

```
import java.net.*;
import java.io.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**
 * Summary description for LoginPage
 *
 */
public class LoginPage extends JFrame
{
                    // Variables declaration
```

VIEW THE MESSAGE BY THE DESTIANTION

APPENDIX B

SAMPLE CODING

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.io.*;

/**
 * Summary description for SecTopology
 *
 */
public class SecTopology extends JFrame
{
        // Variables declaration
        private JLabel lTitle;
        private JLabel lImage;
        private JLabel lHopName;
        private JLabel lHopPass;
        private JLabel lHopCost;
        private JLabel lHst_Name;
        private JLabel lPort;
        private JTextField tfHopNa;
        private JTextField tfHoCos;
        private JTextField tfHo_Name;
        private JPasswordField pfHoPass;
        private JTextField tfPort_Num;
        private JButton bSub;
        private JComboBox cbConList;
        private JButton bLogin;
        private JPanel contentPane;
        private JLabel lConHop;

        String Nodename,pass,cost,ipaddress,portnum,aa="",conval="";
        ResultSet rs;
        Statement st;
        Connection con;
        // End of variables declaration


        public SecTopology()
        {
                super();
                initializeComponent();
                //
        // TODO: Add any constructor code after initializeComponent call
        //

        this.setVisible(true);
        this.setSize(850, 600);
        this.setVisible(true);
        this.setResizable(false);
}


private void initializeComponent()
{
        lTitle = new JLabel();
        lImage = new JLabel();
        lHopName = new JLabel();
        lHopPass = new JLabel();
        lHopCost = new JLabel();
        lHst_Name = new JLabel();
        lPort = new JLabel();
        tfHopNa = new JTextField();
        tfHoCos = new JTextField();
        tfHo_Name = new JTextField();
        pfHoPass = new JPasswordField();
        cbConList = new JComboBox();
        tfPort_Num = new JTextField();
        lConHop = new JLabel();
        bSub = new JButton();
        bLogin = new JButton();
        contentPane = (JPanel)this.getContentPane();
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        //
        // lTitle
        //
        lTitle.setForeground(new Color(153, 0, 153));
        lTitle.setText("Topology Creation");
        //
        // lImage
        //
        lImage.setIcon(new ImageIcon("topology.jpg"));
        //
        // lHopName
        //
        lHopName.setForeground(new Color(153, 0, 153));
        lHopName.setText("Hop Name");
        //
        // lHopPass
```

```
  //
  lHopPass.setForeground(new Color(153, 0, 153));
  lHopPass.setText("Hop Password");
  //
  // lHopCost
  //
  lHopCost.setForeground(new Color(153, 0, 153));
  lHopCost.setText("Hop Cost");
  //
  // lHst_Name
  //
  lHst_Name.setForeground(new Color(153, 0, 153));
  lHst_Name.setText("IP Address");
  //
  // lPort
  //
  lConHop.setForeground(new Color(153, 0, 153));
  lConHop.setText("Connected Hop");
  lConHop.setFont(new Font("Serif", Font.PLAIN, 15));
  //
  lPort.setForeground(new Color(153, 0, 153));
  lPort.setText("Port Number");
  //
  // tfHopNa
  //
  tfHopNa.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e)
          {
                  tfHopNa_actionPerformed(e);
          }

  });
  cbConList.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e)
          {
                  cbConList_actionPerformed(e);
          }

  });
  //
// tfHoCos
  //
  tfHoCos.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e)
          {
                  tfHoCos_actionPerformed(e);
```

```
                }

        });
//
// tfHo_Name
 //         /
        tfHo_Name.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                        tfHo_Name_actionPerformed(e);
                }

        });
        //
        // pfHoPass
        //
        pfHoPass.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                        pfHoPass_actionPerformed(e);
                }

        });
        //
        // tfPort_Num
        //
        tfPort_Num.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                        tfPort_Num_actionPerformed(e);
                }

        });
        //
        // bSub
        //
        bSub.setForeground(new Color(153, 0, 153));
        bSub.setText("Create Topology");
        bSub.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                        bSub_actionPerformed(e);
                }

        });
        //
```

```
		// blogin
	//
		bLogin.setForeground(new Color(153, 0, 153));
		bLogin.setText("Path Selection");
		bLogin.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e)
			{
				bLogin_actionPerformed(e);
			}

		});
		//
		// contentPane
		//
		contentPane.setLayout(null);
		contentPane.setBackground(new Color(255, 255, 255));
		addComponent(contentPane, lTitle, 210,54,160,40);
		addComponent(contentPane, lImage, 480,99,297,369);
		addComponent(contentPane, lHopName, 152,151,98,29);
		addComponent(contentPane, lHopPass, 152,216,108,28);
		addComponent(contentPane, lHopCost, 152,278,90,26);
		addComponent(contentPane, lHst_Name, 152,337,96,30);
		addComponent(contentPane, lPort, 152,460,96,30);
		addComponent(contentPane, tfHopNa, 300,155,100,22);
		addComponent(contentPane, tfHoCos, 300,280,100,22);
		addComponent(contentPane, tfHo_Name, 300,341,100,22);
		addComponent(contentPane, pfHoPass, 300,216,100,22);
		addComponent(contentPane, tfPort_Num, 300,461,100,22);
		addComponent(contentPane, bSub, 127,519,150,35);
		addComponent(contentPane, bLogin, 326,519,150,35);
		addComponent(contentPane, lConHop, 152,397,94,26);
		addComponent(contentPane, cbConList, 300,397,100,22);
		lConHop.setVisible(false);
		cbConList.setVisible(false);
		this.setTitle("SecTopology - extends JFrame");
		this.setLocation(new Point(11, 21));
		this.setSize(new Dimension(653, 518));
	}

		private void addComponent(Container container,Component c,int x,int
y,int width,int height)
	{
		c.setBounds(x,y,width,height);
		container.add(c);
	}
```

```
        //

        private void tfHopNa_actionPerformed(ActionEvent e)
        {
                System.out.println("\ntfHopNa_actionPerformed(ActionEvent e) called.");

        }

        private void tfHoCos_actionPerformed(ActionEvent e)
        {
                System.out.println("\ntfHoCos_actionPerformed(ActionEvent e) called.");

        }

        private void tfHo_Name_actionPerformed(ActionEvent e)
        {
                System.out.println("\ntfHo_Name_actionPerformed(ActionEvent e)
called.");


        }

        private void pfHoPass_actionPerformed(ActionEvent e)
        {
                System.out.println("\npfHoPass_actionPerformed(ActionEvent e)
called.");

        }

        private void tfPort_Num_actionPerformed(ActionEvent e)
        {
                System.out.println("\ntfPort_Num_actionPerformed(ActionEvent e)
called.");


        }
        private void cbConList_actionPerformed(ActionEvent e)
        {
                System.out.println("\ncbConList_actionPerformed(ActionEvent e)
called.");

                Object o = cbConList.getSelectedItem();
                System.out.println(">>" + ((o==null)? "null" : o.toString()) + " is
selected.");
                being selected
                  conval = (String)o;
        }

        private void bSub_actionPerformed(ActionEvent e)
        {
                System.out.println("\nbSub_actionPerformed(ActionEvent e) called.");

                try
                {
                DB();
                Nodename=tfHopNa.getText();
                pass=pfHoPass.getText();
                cost=tfHoCos.getText();
                ipaddress=tfHo_Name.getText();
                portnum=tfPort_Num.getText();
                rs=st.executeQuery("select count(*) from topology");
                while(rs.next())
                {
                        int count=Integer.parseInt(rs.getString(1));
                        System.out.println("Count  :"+count);
                        if(count==0)
                        {
                                st.executeUpdate("insert into topology
values('"+Nodename+"','"+pass+"','"+cost+"','"+ipaddress+"','"+Nodename+"','"+portnum
+"')");
                                JOptionPane.showMessageDialog(this,"Hop Inserted
Sucessfully!!! Hop Name  :  "+Nodename);
                                 lConHop.setVisible(true);
                                cbConList.setVisible(true);
                                cbConList.addItem(Nodename);
                                 tfHopNa.setText("");
                                 pfHoPass.setText("");
                                 tfHoCos.setText("");
                                 tfHo_Name.setText("");
                                 tfPort_Num.setText("");

                        }
                        else if(count>=1)
                        {

        rs=st.executeQuery("select * from topology");
                                while(rs.next())
                                {

                                        aa = rs.getString(5).trim();
                                        System.out.println("FIVETH VALUE  :"+aa);
                                }
                                modify1();

                        }



            }

                  }
                  catch (Exception ee)
                  {
                        ee.printStackTrace();
                  }
                        }
            private void bLogin_actionPerformed(ActionEvent e)
            {
                  System.out.println("\nbLogin_actionPerformed(ActionEvent e) called.");


                  new PathSelection();

            }




public void DB()
        {
         try
         {
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          con=DriverManager.getConnection("Jdbc:Odbc:secure","sa","");
         st=con.createStatement();
         }
         catch (Exception ep)
         {
                ep.printStackTrace();
         }

        }
        public void modify1()
{
 try

 {
       String name,pass,cost,ipval,portVal;
```

```
                name = tfHopNa.getText().trim();
                System.out.println("NAME  :"+name);
                pass = pfHoPass.getText().trim();
                cost = tfHoCos.getText().trim();
                ipval = tfHo_Name.getText().trim();
                portVal = tfPort_Num.getText().trim();
                        String rr = aa + "#" + name;
                System.out.println("Final Value  "+rr);
                cbConList.addItem(name);
                 String ww  = ("insert into topology
values('"+name+"','"+pass+"','"+cost+"','"+ipval+"','"+rr+"','"+portVal+"')");
                st.executeUpdate(ww);
                System.out.println("second insert              "+ww);
      JOptionPane.showMessageDialog(this,"Hop Inserted Sucessfully!!! Hop Name  :
"+Nodename);
                 tfHopNa.setText("");
            pfHoPass.setText("");
                 tfHoCos.setText("");
                 tfHo_Name.setText("");
                 tfPort_Num.setText("");
         }
        catch(Exception f)
        {
                        f.printStackTrace();
        }
}
```

REFERENCES

[1] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, "Securing User-Controlled Routing Infrastructures", IEEE/ACM TRANSACTIONSON NETWORKING, vol. 16, No. 3, June 2008

[2] D. S. Alexander,W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, "A secure active network environment architecture," *IEEE Network*, vol. 12, pp. 37–45, 3, May-Jun. 1998.

[3] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing internet denial- of-service with capabilities," in *Proc. Hotnets*, 2003.

[4] S. Bellovin, "Security concerns for IPng," RFC 1675, 1994.

[5] K. L. Calvert, J. Griffioen, and S. Wen, "Lightweight network support for scalable end-to-end services," in *Proc. ACM SIGCOMM*, 2002.

[6] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *Proc. OSDI*, Dec. 2002.

[7] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," Mar. 1999.

[8] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," in *Proc. 10th USENIX Security Symp.*, 2001.

[9] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology—CRYPTO'92, International Association for Crypto logic Research*, ser. LNCS 740, E. Brickell, Ed. Berlin, Germany: Springer-Verlag, 1993, pp. 139–147.