P-3565

DYNAMIC KEY MANAGEMENT FOR ADHOC NETWORKS

PROJECT REPORT

Submitted by

R.S.MOHANA PRIYA

Reg. No: 0920108009

In partial fulfillment for the award of the degree of

MASTER OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 049

APRIL 2011

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 049

Department of Computer Science and Engineering

PROJECT WORK

PHASE II

APRIL 2011

This is to certify that the project entitled

DYNAMIC KEY MANAGEMNT FOR ADHOC NETWORKS

is the bonafide record of project work done by

R.S.MOHANA PRIYA

Register No: 0920108009

of M.E. (Computer Science and Engineering) during the year 2010-2011.

Project Guide

Head of the Department

Submitted for the Project Viva-Voce examination held on 25-4-2011

DECLARATION

I affirm that the project work titled **"DYNAMIC KEY MANAGEMENT FOR ADHOC NETWORKS"** being submitted in partial fulfillment for the award of M.E. degree is the original work carried out by me. It has not formed the part of any other project work submitted for the award of any degree or diploma, either in this or any other University.

R.S.M

MOHANAPRIYA R.S.
Register No: 0920108009

I certify that the declaration made above by the candidate is true

Mrs. **N.SUGANTHI M.E., (Ph.D.,)**

Associate Professor
Department of Information Technology,
Kumaraguru College of Technology,
(An Autonomous Institution)
Coimbatore-641 049.

Karunya UNIVERSITY

(Karunya Institute of Technology and Sciences)
Declared as Deemed to be University Under sec. 3 of the UGC Act, 1956
Karunya Nagar, Coimbatore 641 114. India

Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that Dr/Mr/Ms/Mrs...R.S.Mohanapriya.....of Kumaraguru College of technology, Coimbatore.....has ~~participated~~ /presented a paper titled..."Energy efficient cluster formation in dynamic sensor networks"..... in the International Conference on "Communication and Signal Processing (ICCOS'11)" on 17th & 18th March 2011 organized by the School of Electrical Sciences, Department of Electronics and Communication Engineering, Karunya University, Coimbatore, India.

Dr. A. Ravi Sankar
Convener

Dr. (Mrs.) Anne Mary Fernandez
Patron

Dr. Paul P. Appasamy
Patron

onal Conference on Compu

NCCCIS-2011

February

Organized by

Department of Information Technology

CERTIFICATE

is certified that R.S. MOHANAPRIYHA of KUMARAGURU COLLEGE OF

ted a paper titled EFFICIENT AND DYNAMIC KEY MANAGEMENT FOR SENSOR NETWORKS

ional Conference on Computing, Communication and Information Systems (NCCCIS-2011) held at

a College of Engineering and Technology, Coimbatore on February 11-12, 2011.

N.V.L.L

Principal

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for enabling me to complete this project.

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

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

I take this opportunity to thank **Mrs.P.Devaki M.E., (Ph.D),** *Head of the Department,* Computer Science and Engineering, for his precious suggestions.

I thank all project committee members for their comments and advice during the reviews. Special thanks to **Mrs.V.Vanitha M.E.,** *Associate Professor*, Department of Computer science and Engineering, for arranging the brain storming project review sessions.

I register my hearty appreciation to the Guide **Mrs.N.Suganthi M.E.,** *Associate Professor,* Department of Information Technology, my thesis advisor. I thank for her support, encouragement and ideas. I thank her for the countless hours she has spent with me, discussing everything from research to academic choices.

I would like to convey my honest thanks to all **Teaching** staff members and **Non Teaching** staffs of the department for their support. I would like to thank all my classmates who gave me a proper light moments and study breaks apart from extending some technical support whenever I needed them most.

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

ABSTRACT

Wireless Sensor Networks (WSNs) which is a type of ad-hoc network present a challenging secure environment because of their limited resources, heterogeneity, and highly dynamic nature. Since the network is dynamic, security is a major threat in such cases. Key management is an important cryptographic technique for providing security in a dynamic environment of sensor networks. Existing key management schemes are not much suitable for sensor networks. They are not efficient for a dynamic topology, or link failures. In this project pair wise key generation schemes and group key generation schemes of existing schemes are implemented. Then proposed method of securely generating pair wise and group key scheme is done, where if the communication is between two nodes pair wise keys are generated and if the communication is among group of nodes group keys are generated. The proposed method is compared with the existing method in various issues like computation, communication and memory complexity.

ஆய்வுச்சுருக்கம்

கம்பியில்லா உணர்வி இணையமானது ஒரு வகையான குறித்த காரியத்திற்காக அமைக்கப்பட்ட பிணையம். அவ்வகை கணினி கட்டமைப்புக்கு சவாலாக பல சூழ்நிலைகள் அமைகிறது. அதன் மாறும் நிலை, அளவான ஆதாரங்கள் ஆகியவை பாதுகாப்புக்கு மிகுந்த அச்சுறுத்தலை விளைவிக்கிறது. விசை மேலாண்மை என்னும் ஒரு வகையான மறையீட்டு நுணுக்கமுறையை உபயோகித்து பாதுகாப்பான கணினி சூழலலை உருவாக்கலாம். நடைமுறையில் உள்ள திட்டமுறைகள் தகுந்த ஆற்றலை தருவதில்லை. நடைமுறையில் உள்ள தொழில்நுட்ப முறைகள் தொடர்பு இணைப்பு செயலிழப்பு,மாறும் நிலை நிலவுருவியல் போன்ற காரணங்களினால் சிறந்த ஆற்றலை தருவதில்லை.எனவே இந்த ஆய்வில் இணை விசை மற்றும் குழு விசை முறைகள் அமல்படுத்தப்பட்டது.முற்போத்தல் செயல்முறையில் இணை விசை,குழு விசை ஆகியவை பாதுகாப்பான முறையில் செயல்படுத்தப்பட்டது. இரு முடிச்சுகள் இடையே பாதுகாப்பான தொடர்புக்கு இணை விசையும், குழு முடிச்சுகள் இடையே பாதுகாப்பான தொடர்புக்கு குழு விசையும் செயல்படுத்தப்பட்டது. இதில் கணணித்துவ சிக்கல் ,தொடர்பு உட்சிக்கல் நிலை போன்ற பல இடவுகள் ஒப்பிடப்பட்டுள்ளது.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
MANET	Mobile Ad-hoc NETwork
CRP	Constrained Random Perturbation
WSN	Wireless Sensor Network
GC	Group Controller
GF	Galois Field
JFC	Java Foundation Class
GDH	Group Diffie Hellman
PDF	Probability Density Function

CHAPTER 1
INTRODUCTION

1.1 WIRELESS AD-HOC NETWORKS

1.1.1 General:

A wireless ad-hoc network is a collection of mobile/semi-mobile nodes with no pre established infrastructure, forming a temporary network. Each of the nodes has a wireless interface and communicates with each other over either radio or infrared. Laptop computers and personal digital assistants that communicate directly with each other are some examples of nodes in an ad-hoc network. Nodes in the ad-hoc network are often mobile, but can also consist of stationary nodes, such as access points to the Internet. Semi mobile nodes can be used to deploy relay points in areas where relay points might be needed temporarily [1].

Figure 1 shows a simple ad-hoc network with three nodes. The outermost nodes are not within transmitter range of each other. However the middle node can be used to forward packets between the outermost nodes. The middle node is acting as a router and the three nodes have formed an ad-hoc network.

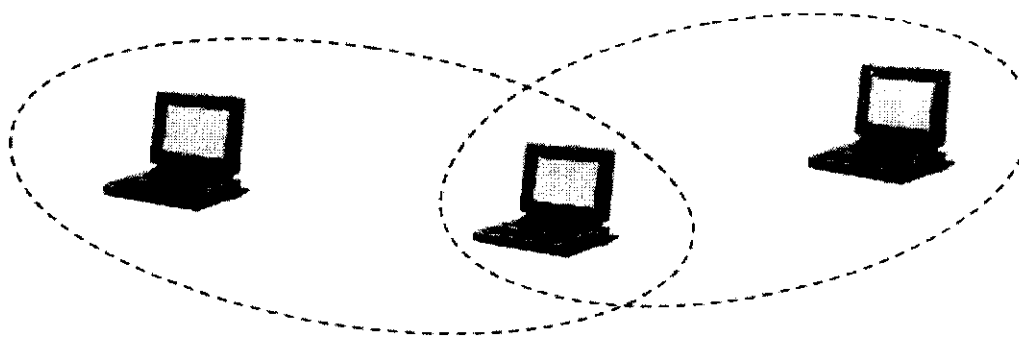


Figure 1: Example of a simple ad-hoc network with three participating nodes.

An ad-hoc network uses no centralized administration. This is to be sure that the network won't collapse just because one of the mobile nodes moves out of transmitter range of the others. Nodes should be able to enter/leave the network as they wish. Because of the limited transmitter range of the nodes, multiple hops may be needed to reach other nodes. Every node wishing to participate in an ad hoc network must be willing to forward packets for other nodes. Thus every node acts both as a host and as a router. A node can be viewed as an abstract entity consisting of a router and a set of affiliated mobile hosts (Figure 2). A router is an entity, which, among other things runs a routing protocol [1].

Ad-hoc networks are also capable of handling topology changes and malfunctions in nodes. It is fixed through network reconfiguration. For instance, if a node leaves the network and causes link breakages, affected nodes can easily request new routes and the problem will be solved. This will slightly increase the delay, but the network will still be operational.

Wireless ad-hoc networks take advantage of the nature of the wireless communication medium. In other words, in a wired network the physical cabling is done a priori restricting the connection topology of the nodes. This restriction is not present in the wireless domain and, provided that two nodes are within transmitter range of each other, an instantaneous link between them may form.

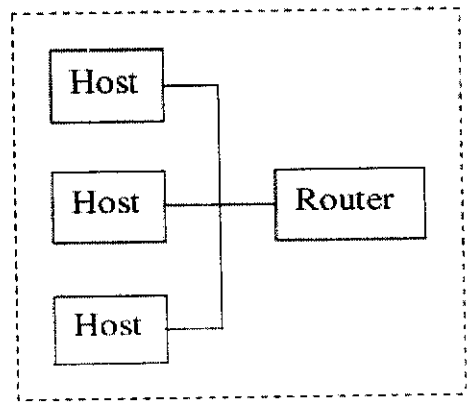


Figure 2: Block diagram of a mobile node acting both as hosts and as router.

1.1.2 Usage:

There is no clear picture of what these kinds of networks will be used for. The suggestions vary from document sharing at conferences to infrastructure enhancements and military applications .In areas where no infrastructure such as the Internet is available an ad-hoc network could be used by a group of wireless mobile hosts. This can be the case in areas where a network infrastructure may be undesirable due to reasons such as cost or convenience. Examples of such situations include disaster recovery personnel or military troops in cases where the normal infrastructure is either unavailable or destroyed.

Other examples include business associates wishing to share files in an airport terminal, or a class of students needing to interact during a lecture. If each mobile host wishing to communicate is equipped with a wireless local area network interface, the group of mobile hosts may form an ad-hoc network. Access to the Internet and access to resources in networks such as printers are features that probably also will be supported.

1.1.3 Characteristics:

Ad-hoc networks are often characterized by a dynamic topology due to the fact that nodes change their physical location by moving around. This favors routing protocols that dynamically discover routes over conventional routing algorithms like distant vector and link state. Another characteristic is that a host/node has very limited CPU capacity, storage capacity, battery power and bandwidth, also referred to as a "thin client". This means that the power usage must be limited thus leading to a limited transmitter range.

The access media, the radio environment, also has special characteristics that must be considered when designing protocols for ad-hoc networks. One example of this may be unidirectional links. These links arise when for example two nodes have different strength on their transmitters, allowing only one of the host to hear the other, but can also arise from disturbances from the surroundings. Multihop in a radio environment may result in an overall transmit capacity gain and power gain, due to the squared relation between coverage and required output power. By using multihop, nodes can transmit the packets with a much lower output power.

Thus, ad hoc networks have several salient characteristics [5]:

- dynamic topologies
- resource constraints
- limited physical security
- no infrastructure

Wireless ad hoc networks can be further classified by their application:

- mobile ad hoc networks (MANET)
- wireless mesh networks (WMN)
- wireless sensor networks (WSN)

1.2 WIRELESS SENSOR NETWORKS:

A Wireless Sensor Network (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants and to cooperatively pass their data through the network to a main location. The more modern networks are bi-directional, enabling also to control the activity of the sensors. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance; today such networks are used in many industrial and consumer application, such as industrial process monitoring and control, machine health monitoring, environment and habitat monitoring, healthcare applications, home automation, and traffic control [1].

The WSN is built of "nodes" - from a few to several hundreds or even thousands, where each node is connected to one (or sometimes several) sensors. Each such sensor network node has typically several parts: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery. A sensor node might vary in size from that of a shoebox down to the size of a grain of dust, although functioning "motes" of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from hundreds of dollars to a few pennies, depending on the complexity of the individual sensor nodes.

Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communications bandwidth.

1.2.1 Applications:

1.2.1.1 Area monitoring

Area monitoring is a common application of WSNs [1]. In area monitoring, the WSN is deployed over a region where some phenomenon is to be monitored. A military example is the use of sensors to detect enemy intrusion; a civilian example is the geo-fencing of gas or oil pipelines. When the sensors detect the event being monitored (heat, pressure), the event is reported to one of the base stations, which then takes appropriate action (e.g., send a message on the internet or to a satellite). Similarly, wireless sensor networks can use a range of sensors to detect the presence of vehicles ranging from motorcycles to train cars.

1.2.1.2 Environmental monitoring

A number of WSNs have been deployed for environmental monitoring.

Air pollution monitoring

Wireless sensor networks have been deployed in several cities (Stockholm, London or Brisbane) to monitor the concentration of dangerous gases for citizens. The sensor nodes can control important parameters like CO, CO2, NO2 or CH4, which are generated by vehicles and industry, and have a severe impact on the human health. This way, the public institutions have a good tool to design plans to reduce pollution, improve the air quality and ensure the compliance with current legislation.

Forest fire detection

A network of Sensor Nodes can be installed in a forest to control when a fire has started. The nodes will be equipped with sensors to control temperature, humidity and gases which are produced by fire in the trees or vegetation.

The early detection is crucial for a successful action of the firefighters; thanks to Wireless Sensor Networks, the fire brigade will be able to know when a fire is started and how it is spreading.

Greenhouse monitoring

Wireless sensor networks are also used to control the temperature and humidity levels inside commercial greenhouses. When the temperature and humidity drops below specific levels, the greenhouse manager must be notified via e-mail or cell phone text message, or host systems can trigger misting systems, open vents, turn on fans, or control a wide variety of system responses.

Landslide detection

A landslide detection system makes use of a wireless sensor network to detect the slight movements of soil and changes in various parameters that may occur before or during a landslide. And through the data gathered it may be possible to know the occurrence of landslides long before it actually happens.

1.2.2 Challenges of Sensor Security

A wireless sensor network is a special network which has many constraints compared to a traditional computer network. Due to these constraints it is difficult to directly employ the existing security approaches to the area of wireless sensor networks. Therefore, to develop useful security mechanisms while borrowing the ideas from the current security techniques, it is necessary to know and understand these constraints first.

Very Limited Resources

All security approaches require a certain amount of resources for the implementation, including data memory, code space, and energy to power the sensor. However, currently these resources are very limited in a tiny wireless sensor.

• Limited Memory and Storage Space: A sensor is a tiny device with only a small amount of memory and storage space for the code [5].

In order to build an effective security mechanism, it is necessary to limit the code size of the security algorithm. For example, one common sensor type (TelosB) has an 16-bit, 8 MHz RISC CPU with only 10K RAM, 48K program memory, and 1024K flash storage. With such a limitation, the software built for the sensor must also be quite small. The total code space of TinyOS, the de-facto standard operating system for wireless sensors, is approximately 4K, and the core scheduler occupies only 178 bytes. Therefore, the code size for the all security related code must also be small.

• Power Limitation: Energy is the biggest constraint to wireless sensor capabilities. We assume that once sensor nodes are deployed in a sensor network, they cannot be easily replaced (high operating cost) or recharged (high cost of sensors). Therefore, the battery charge taken with them to the field must be conserved to extend the life of the individual sensor node and the entire sensor network. When implementing a cryptographic function or protocol within a sensor node, the energy impact of the added security code must be considered. When adding security to a sensor node, we are interested in the impact that security has on the lifespan of a sensor (i.e., its battery life). The extra power consumed by sensor nodes due to security is related to the processing required for security functions (e.g., encryption, decryption, signing data, verifying signatures), the energy required to transmit the security related data or overhead (e.g., initialization vectors needed for encryption/decryption), and the energy required to store security parameters in a secure manner (e.g., cryptographic key storage).

Unreliable Communication

Certainly, unreliable communication is another threat to sensor security. The security of the network relies heavily on a defined protocol, which in turn depends on communication.

• Unreliable Transfer normally the packet-based routing of the sensor network is connectionless and thus inherently unreliable. Packets may get damaged due to channel errors or dropped at highly congested nodes. The result is lost or missing packets. Furthermore, the unreliable wireless communication channel also results in damaged packets. Higher channel error rate also forces the software developer to devote resources to error handling.

More importantly, if the protocol lacks the appropriate error handling it is possible to lose critical security packets. This may include, for example, a cryptographic key.

• Conflicts: Even if the channel is reliable, the communication may still be unreliable. This is due to the broadcast nature of the wireless sensor network. If packets meet in the middle of transfer, conflicts will occur and the transfer itself will fail. In a crowded (high density) sensor network, this can be a major problem. More details about the effect of wireless communication can be found.

• Latency: The multi-hop routing, network congestion and node processing can lead to greater latency in the network, thus making it difficult to achieve synchronization among sensor nodes. The synchronization issues can be critical to sensor security where the security mechanism relies on critical event reports and cryptographic key distribution.

Unattended Operation

Depending on the function of the particular sensor network, the sensor nodes may be left unattended for long periods of time. There are three main caveats to unattended sensor nodes:

• Exposure to Physical Attacks: The sensor may be deployed in an environment open to adversaries, bad weather, and so on. The likelihood that a sensor suffers a physical attack in such an environment is therefore much higher than the typical PCs, which is located in a secure place and mainly faces attacks from a network.

• Managed Remotely: Remote management of a sensor network makes it virtually impossible to detect physical tampering (i.e., through tamperproof seals) and physical maintenance issues (e.g., battery replacement). Perhaps the most extreme example of this is a sensor node used for remote reconnaissance missions behind enemy lines. In such a case, the node may not have any physical contact with friendly forces once deployed.

• No Central Management Point: A sensor network should be a distributed network without a central management point. This will increase the vitality of the sensor network. However, if designed incorrectly, it will make the network organization difficult, inefficient, and fragile.

1.3 Defining Key Management:

A keying relationship is the state wherein network nodes share keying material for use in cryptographic mechanisms. The keying material can include public/private key pairs, secret keys, initialization parameters, and non-secret parameters supporting key management in various instances. Key management can be defined as a set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties [13].

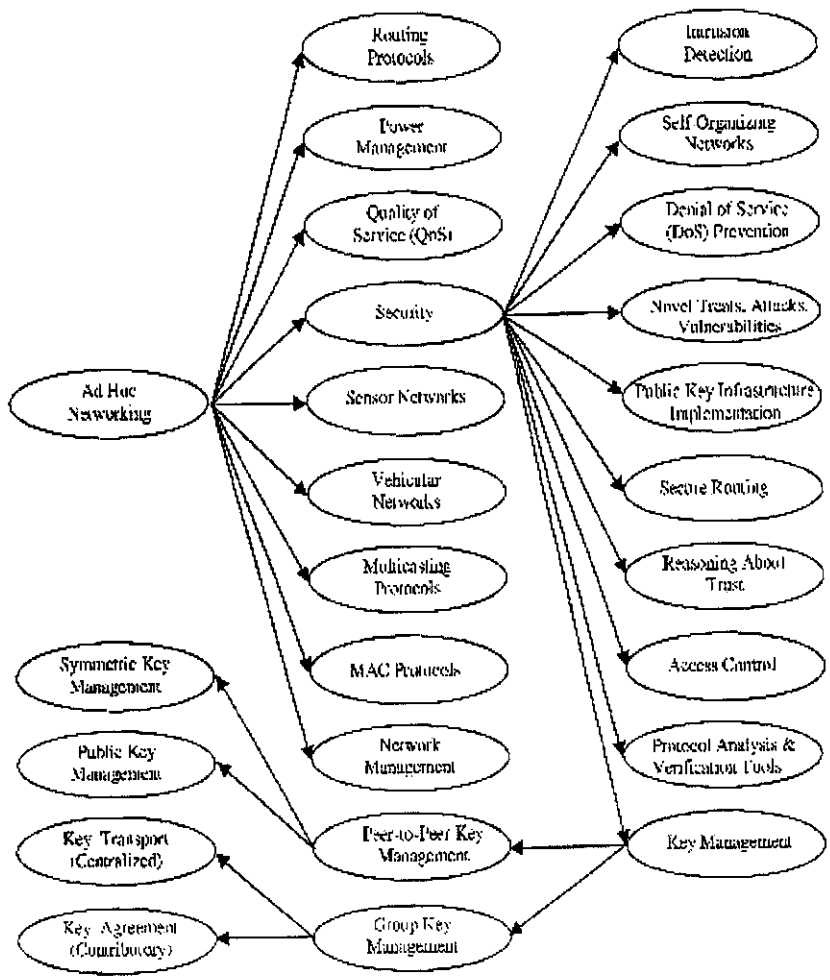


Figure 3: Dimensions of ad-hoc networks.

In summary, key management integrates techniques and procedures to establish a service supporting [13]:

(1) Initialization of system users within a network;

(2) Generation, distribution, and installation of keying material;

(3) control over the use of keying material;

(4) update, revocation, and destruction of keying material;

(5) Storage, backup/recovery, and archival of keying material, and

(6) Bootstrapping and maintenance of trust in keying material.

Authentication is the basis of secure communication [15]. Without a robust authentication mechanism in place, the remaining security goals (confidentiality, data integrity, and nonrepudiation) are in most instances not achievable. Authentication can only be realized by means of verifying something known to be associated with an identity. In the electronic domain, the owner of the identity must have a publicly verifiable secret associated with its identity; otherwise, the node can be impersonated. Authentication in general depends on the context of usage. Key management is concerned with the authenticity of the identities associated with the six services given above; it is a concept which may seem trivial at first, but one that is not easily achieved. Authentication of users is particularly difficult (and in most network settings impossible) without the help of a trusted authority.

The fundamental function of key management schemes is the establishment of keying material [15]. Key establishment can be subdivided into key agreement and key transport. Key agreement allows two or more parties to derive shared keying material as a function of information contributed by, or associated with, each of the protocol participants, such that no party can predetermine the resulting value. In key transport protocols, one party creates or otherwise obtains keying material, and securely transfers it to the other party or parties. Both key agreement and key transport can be achieved using either symmetric or asymmetric techniques. A hybrid key establishment scheme makes use of both symmetric and asymmetric techniques in an attempt to exploit the advantages of both techniques.

1.3.1 Requirements of Key Management Schemes:

Key management services should adhere to the following generic security attributes [15]:

—Confidentiality. The protection of data from unauthorized disclosure is called confidentiality. It includes connection confidentiality, connectionless confidentiality, selective-field confidentiality and traffic flow confidentiality.

—Key authentication. Key authentication is a property whereby an assurance is given that the communicating entity is the one that it claims to be. It includes peer entity authentication and data origin authentication.

Key authentication, in the context of a communication session between two parties, can either be unilateral or mutual: unilateral authentication means that only one party's keying material is authenticated, while mutual authentication involves validating both parties keying material. Possession of the key is in fact independent of key authentication. Key authentication, without knowledge that the intended recipient actually has the relevant key, is referred to as implicit key authentication.

—Key confirmation. If key confirmation is provided by a key establishment protocol, communication entities prove possession of authenticated keying material. Key authentication with key confirmation yields explicit key authentication.

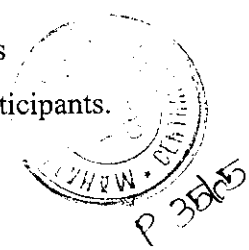
—Key freshness. The key freshness property improves security by ensuring new and independent keys between different communication sessions. By separating communication sessions, the available information for cryptanalytic purposes is limited, which makes cryptanalytic attack more difficult.

—Perfect forward secrecy. Perfect forward secrecy (PFS) ensures that compromise of long-term keys cannot result in compromise of past session keys.

—Resistant to known key attacks. A key management scheme is vulnerable to known key attacks (KKA) if a compromised past session key or subset of past session keys allows the following:

(1) a passive adversary to compromise future session keys

(2) An active adversary to impersonate other protocol participants.



—Forward secrecy. A key management scheme with a forward secrecy property prevents an adversary from discovering subsequent keys from a compromised contiguous subset of old keys.

—Backward secrecy. A key management scheme with a backward secrecy property prevents an adversary from discovering preceding keys from a compromised contiguous subset of old keys.

—Key independence. Key independence guarantees that a passive adversary who knows a proper subset of keys cannot discover any other keys. Key independence subsumes forward and backward secrecy. Key independence does not imply key freshness.

—Availability. A high-availability feature prevents degradation of key management services and ensures that keying material is provided to nodes in the network when expected.

—Robustness. The key management scheme should tolerate hardware and software failures, asymmetric and unidirectional links, and network fragmentation/partitioning due to limited/error prone wireless connectivity.

—Survivability. Survivability is the capability of the key management service to re main available even in the presence of threats and failures. Survivability goes beyond security and fault tolerance to focus on the delivery of services, even when the system is partly compromised or experiences failures. (Survivability thus subsumes robustness.) Rapid recovery of services is required when conditions improve. Survivability includes byzantine robustness, which implies that the key management service should be able to function properly even if some misbehaving participating nodes attempt to disrupt its operation.

CHAPTER 2
LITERATURE SURVEY

2.1 An optimal class of symmetric key generation systems

Bloms scheme is a classical threshold based key distribution scheme. The keys shared between users are distributed at start up time and it is called key generation authority and using which session keys are generated [3].

A network with n users implies that each user must have access to n-1 keys. Now, if n is large it becomes impractical or even impossible to store all keys securely. A natural solution would then be to supply each user with a relatively small amount of secret data from which he can derive all his keys. This is called symmetric key generation system (SKGS)[3].However, if all keys are generated from a small amount of data attention must be payed to the fact that dependencies between keys will exist. These dependencies will be such that a group of cooperating users might be able to decrease their uncertainty about keys they should not know about.

2.1.1 Generating secret key:

The users in the system are numbered consequtively from 1 to n. Also assume that at least k users shall have to cooperate to get any information about a key they should not have access to. We also assume that the keys should be in GF (q).The construction of the SKGS (Symmetric Key Generation System) starts with selecting a(n,k) MDS code over GF(q) with generator matrix G. This G will be known by all users in the network. Then the key generating authority draws a random symmetric matrix D, also with elements in GF (q). The keys to be used by the user pairs are then given by $K= (DG)^T.G$ [3]

User pair (i,j) will use $(k)_{i,j}$ i.e. the element in row i and column j in K. Obviously K is symmetric and hence $(K)_{i,j}=(K)_{j,i}$. Then if user i knows row i of K and user j knows row j of K, they have a common key. The i^{th} row in K is given by the ith row in $(DG)^T$ and G. But G is assumed publicly known so the only data that the key generation authority has to distribute to user i is the i^{th} row of $(DG)^T$.

$(DG)^T$ is a n×k matrix which means that each row consists of k elements in GF(q). Thus the required secret store at each user is k×lb(q) bits. This is really the least possible value for a SKGS with the assumed parameters.

At last they had given a simple explanation of why at least k users have to cooperate to get any information about keys they do not have. Assume m<k users cooperate. Then they know m rows of k. But K is symmetric and then they also know m columns. This means that they know m elements in each row of k for all other users. They do not know any other elements in these rows [3].

Now observe that each row in k is a codeword in the code generated by G. Knowledge of less than k elements in a codeword does not reveal any information about any other element in the codeword. So if less than k users cooperate they get no information about an unknown key. However, if k or more users cooperate they know all of k, because knowledge of k elements in a codeword uniquely determines the codeword [3].

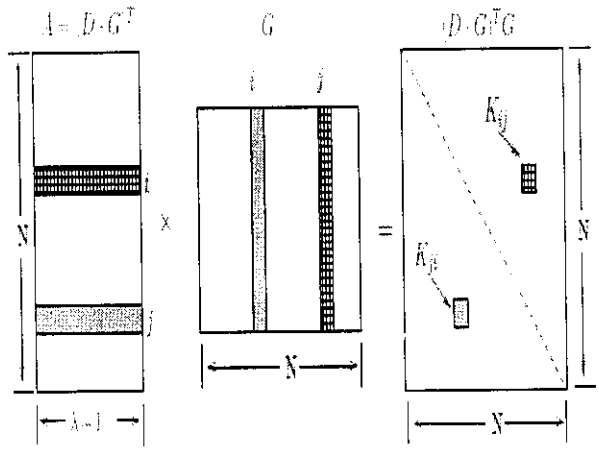


Figure 4: Generating keys in Blom's scheme

2.2 Non interactive Pair wise Key Establishment for Sensor Networks:

They assume that the network consists of N sensor nodes with IDs, $I=\{s_1,s_2,\ldots,s_N\}$ and $s_1<s_2<\ldots,s_N$. Here,$s_1,s_2,\ldots,s_N$ instead of 1,2,…,N , are used as the notations of IDs, which emphasizes that the IDs can be arbitrarily chosen.

They also assume that $q>s_N$, where q is a parameter of a finite field Fq , and ? is an appropriate security parameter independent of N, which leverages the security level and storage overhead[14].

2.2.1 Basic Idea of CARPY:

In Blom's scheme [3], communications become insecure after more than ? sensor nodes are compromised. The reason for this is that the row vector in the Ai,_ in the sensor node *i* is directly related to the private matrix D . Hence, after collecting a sufficient number of row vectors of A, the adversary is able to construct the private matrix D by solving a system of linear equations since G is publicly known. An idea to enhance the security is to break the direct relation between D and A by adding certain random noise on A to distort Blom's key. However, if improper random noise is applied, either additional computation and communication are needed to extract the common bits of distorted Blom's key between two sensor nodes, or the common key cannot be found anymore. To conquer these drawbacks, they propose to apply *constrained random perturbation* (CRP) [14].

The constrained random perturbed Blom's key, when compared to the original Blom's key, will satisfy high signal-to-noise ratio (SNR), i.e., if the length of Blom's key is l , then only the least $r(r<l)$ bits of Blom's key are perturbed after the CRP is added. Thus, the first l-r bits of Blom's key are retained, resulting in the guaranteed establishment of the common key without the need of additional overhead. In contrast to the random perturbation that incurs unnecessary computation and communication overhead, the way of constructing CRP and the corresponding efficiency gain substantially differentiate this work [14]. The main idea of CARPY is shown in Fig. 5. Obviously, the execution of each round of the CARPY scheme can generate l-r bits of a pairwise key. When the bit-length of desired key is $L>(l-r)$, the CARPY scheme should be executed $+_{L/(l-r)}+$ rounds to generate a pairwise key with desired key length. Albeit $+_{L/(l-r)}+$ rounds of CARPY are required. There are two steps in the CARPY scheme, the off-line step and the on-line step [14]. In general, the off-line step is performed, before deployment of sensor nodes, to determine the desired key length, select appropriate parameters, and preinstall the keying materials into the sensor nodes.

The on-line step is performed for each pair of sensor nodes required to find the pairwise key in common after sensor nodes are deployed.

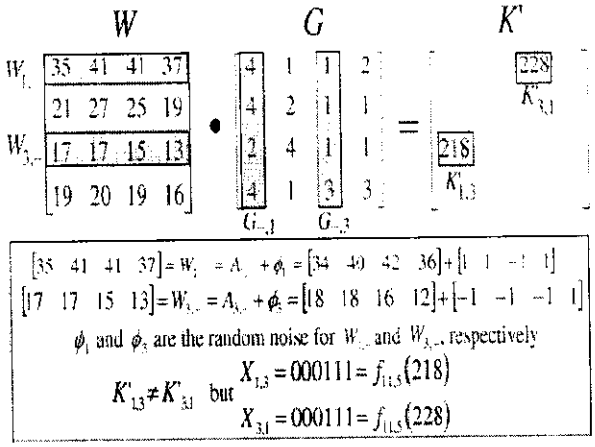


Figure 5: Illustration of CARPY scheme

2.3 Key management scheme for Distributed sensor networks:

A simple key pre-distribution scheme, it requires memory storage for only few tens to a couple of hundred keys. And yet has similar security and superior operational properties when compared to those of the pair-wise private key sharing scheme. This scheme relies on probabilistic key sharing among the nodes of a random graph and uses a simple shared-key discovery protocol for key distribution, revocation and node re-keying. Prior to DSN deployment, they distribute a ring of keys to each sensor node, each key ring consisting of randomly chosen k keys from a large pool of P keys, which is generated off-line. Because of the random choice of keys on key rings, a shared key may not exist between some pairs of nodes. Although a pair of nodes may not share a key, if a path of nodes sharing keys pair-wise exists between the two nodes at network initialization, the pair of nodes can use that path to exchange a key that establishes a direct link. Therefore, full shared-key connectivity offered by a pair-wise private key sharing between every two nodes become unnecessary [6].

2.4 Efficient pair wise key establishment and management scheme

In EPKEM, pair wise communication key is established through four phases: setup key pre-assignment phase, common keys discovery phase, pair wise key computation phase, and key ring establishment phase [6].

2.4.1 Setup Key Pre-Assignment Phase

In setup key pre-assignment phase, the Key Distribution Server (KDS) generates a very large size key pool P with more than 2^{20}distinct keys inside. For each sensor node Ni (Ni denotes this node's ID), KDS randomly selects a secret key from P and stores it into Ni's memory. This key, denoted as K_{Ni+KDS} , is only shared by the KDS and the intended node N_i , and will be used to identify node N_i and secure the communication between N_i and KDS in the future.

Then, for each sensor node, KDS randomly selects a subset keys from the rest of P and pre-loads them into the node's memory, these keys will be worked as the network setup keys after deployment. The KDS assigns setup keys to each node under certain rules to ensure any two nodes have at least two keys in common [6].

ID	1	2	3	4	5	...	m
1	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	...	$k_{1,m}$
2	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	...	$k_{2,m}$
3	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	...	$k_{3,m}$
4	$k_{4,1}$	$k_{4,2}$	$k_{4,3}$	$k_{4,4}$	$k_{4,5}$	...	$k_{4,m}$
5	$k_{5,1}$	$k_{5,2}$	$k_{5,3}$	$k_{5,4}$	$k_{5,5}$	...	$k_{5,m}$
:	:	:	:	:	:	...	:
m	$k_{m,1}$	$k_{m,2}$	$k_{m,3}$	$k_{m,4}$	$k_{m,5}$	...	$k_{m,m}$

Figure 6: An example of setup key matrix K

ID	Keys in key chain
$kc_{1,1}$	$\{k_{1,1},\ldots,k_{1,m},k_{2,1},\ldots,k_{m,1},k_{N_{1,1}\oplus KDS}\}$
:	:
$kc_{i,j}$	$\{k_{i,1},\ldots,k_{i,m},k_{1,j},\ldots,k_{m,j},k_{N_{i,j}\oplus KDS}\}$
:	:
$kc_{m,m}$	$\{k_{m,1},\ldots,k_{m,m},k_{1,m},\ldots,k_{m\oplus 1,m},k_{N_{m,m}\oplus KDS}\}$

Table 1: Keys in key chain

Table 1 lists the constructed key chains, it easy to see that any two key chains $kc_{i,j}$, and $kc_{l,m}$, will share exactly two common keys . Or, they may share m keys in common when i = l or j =m .

2.4.2 Common Keys Discovery Phase

After deployment, any two neighboring sensor nodes need to figure out their shared common keys. To achieve this, each node broadcasts its node ID and key chain ID to its neighbors. Once a node received its neighbor's key chain ID, it can identify which keys they shared in common [6].

2.4.3. Pair wise Key Computation Phase

After the common key discovery phase, each sensor node knows its neighbor node's ID and their shared common keys. Since all the pre-loaded setup keys are picked from the same key matrix K, the same key may be stored in different nodes. That means, when some nodes are captured, keys stored in non-captured nodes may be compromised too. To address this problem, they establish a new pair wise communication key for each pair of neighbor nodes instead of using the shared common keys directly. The new pair wise communication key can be calculated based on the shared setup keys. Suppose node N_a and N_b are a pair of neighbor nodes and their shared setup keys are $K_{i,m}$, and $K_{l,j}$ [6]. To establish a private pair wise key which is unaware to other nodes, node N_a and node N_b compute their pair wise key using Equation 1.

$$k_{N_a \oplus N_b} = k_{i,m} \oplus N_a \oplus k_{l,j} \oplus N_b \qquad (1)$$

In Equation (1), $\oplus$ is the exclusive-or operator. In this scheme, all setup keys in the key matrix K are distinct to each other; any pair of nodes will not share the same two keys as other pairs. Furthermore, each node in the sensor network has a unique identity; hence the calculated pair wise communication key for each pair of sensor nodes is distinct to others and only shared between them. This unique pair wise communication key cannot be computed or guessed by any other nodes, and will be used for data exchange and node authentication between intended neighboring nodes in the network.

2.4.4 Key Ring Establishment phase

Once a sensor node computed all corresponding pair wise communication keys with its neighbors, it erases all the pre assigned setup keys from its memory immediately to prevent the possible key compromising and node capture attack.

Only the computed pair wise communication keys with its neighbors and the secret key shared with KDS are kept in the memory of each node, which compose the permanent key ring of a sensor node. A connected secure link network can be established when the above four phases are finished. Each sensor node can communicate with KDS [6] and the calculated pair wise keys to authenticate and communicate with its proper neighbor nodes securely.

2.5 Random key pre distribution scheme using probability density function:

Here, let's see about an advanced key pre-distribution scheme. The Du et al. scheme [8] uses deployment rectangles whose sizes strongly depend on the PDF of node deployment. It uses a key-position map *and* the probability density function (PDF) of node deployment. The key position map shows which key is assigned to which position [12]; this is specified by coordinates in a two-dimensional coordinate system. Although we can easily extend the concept of the key-position map to three dimensions (including height).The PDF of node deployment can be determined by physical laws or previous results. Similar to the Eschenauer-Gligor scheme, this scheme consists of three phases. The last two phases are the same as in the Eschenauer-Gligor scheme [10]. Therefore, it was focused on the ?rst phase: the key pre-distribution phase.

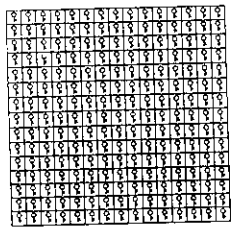


Figure 7: key position map

2.5.1 Prerequisite:

If nodes are deployed from air as described above, their resident points on the ground vary widely because of air re-sistance. This unevenness of the deployment can be modeled using PDFs. In our scheme, we assume that the PDFs of the nodes are known or can be estimated. For example, the PDF Pr(x, y) is a two-dimensional normal distribution. In addition, this scheme employs the communication range of the nodes, denoted as r. For simplicity, we assume that r is a constant.

2.5.2 Key Pre-Distribution Phase:

1. Set security parameters n, m. The parameter n is the size of the key pool and m is the number of keys stored by each node. These parameters are determined according to the resources of the nodes and network resilience.
2. Divide the target deployment area into n areas of the same size. They refer to each area as a subarea. The shape of subareas can be rectangular, square, hexagonal, etc. For simplicity sub areas can be used as rectangles.
3. Generate n keys.
4. A key-position map is created by assigning n keys to n *subareas* in a one-to-one manner. Fig.7 shows an example of the resulting key-position map.

Key distribution to each node [12]

5. For a node S, select one expected resident point P according to the PDF of S, Pr(x, y).
6. Randomly select one point Q within a circle of radius r and center P.
7. Let A *be* the subarea that includes Q. Store the key assigned to A (in step 4) to node S. If the same key already exists in S, start again from step 5.

8. Repeat steps 5–7 until node *S has m keys*.
9. Repeat steps 5–8 for all nodes.

2.6 Polynomial based dynamic key generation:

Polynomial based key pre-distribution scheme distributes a polynomial share (a partially evaluated polynomial) to each sensor node by using which every pair of nodes can generate a link key [4]. Symmetric polynomial P(x, y) (P(x, y) = P(y, x)) of degree ? is used. The coefficients of the polynomial come from GF (q) for sufficiently large prime q. Each sensor node stores a polynomial with ? + 1 coefficient which come from GF (q). Sensor node S_i receives its polynomial share of $f_i(y) = P(i, y)$. S_i (resp. S_j) can obtain link key $K_{i,j} = P(i, j)$ by evaluating its polynomial share $f_i(y)$ (resp. $f_j(y)$) at point j (resp. i). Every pair of sensor nodes can establish a key. The solution is ?-secure; meaning that coalition of less than ?+1 sensor nodes knows nothing about pair-wise keys of others [4].

CHAPTER 3
METHODOLOGY

3.1 Drawbacks of existing approach:

In existing approach, the private key matrix which is kept secret is multiplied directly with the public vector or matrix. Such method is not much secure because linear equations can be formulated and by solving those equations the secret matrix could be obtained. In some of the schemes, a random noise is used to disturb the direct relation between the private secret matrix and the public vector.

By doing so the security can be increased but generating appropriate random noise is a difficult task and has computation overhead. The other methods need more complex calculations but the sensor nodes have limited processing power, and hence it won't be much efficient.

3.2 Proposed work:

In my proposed scheme, the direct use of matrix is not done and another linear independent matrix is introduced and certain computations are done to generate another second level secret matrix in a Galois finite field (GF (q)).An authentication code is also used which is kept secure for the nodes in the network and the nodes are assumed to be tamper proof. Using all these we can generate the pair-wise secret keys.

For generating group keys we can consider a different scenario where the matrix is preloaded and it can be deployed in nodes. Using that matrix we can generate an authentication value and a resultant value which is used to generate the group key. Since the network is dynamic or it changes often, rekeying should be done to provide forward and backward secrecy. A new matrix is generated for this purpose and is sent along with a hello message by the group controller node.

3.3 System model:

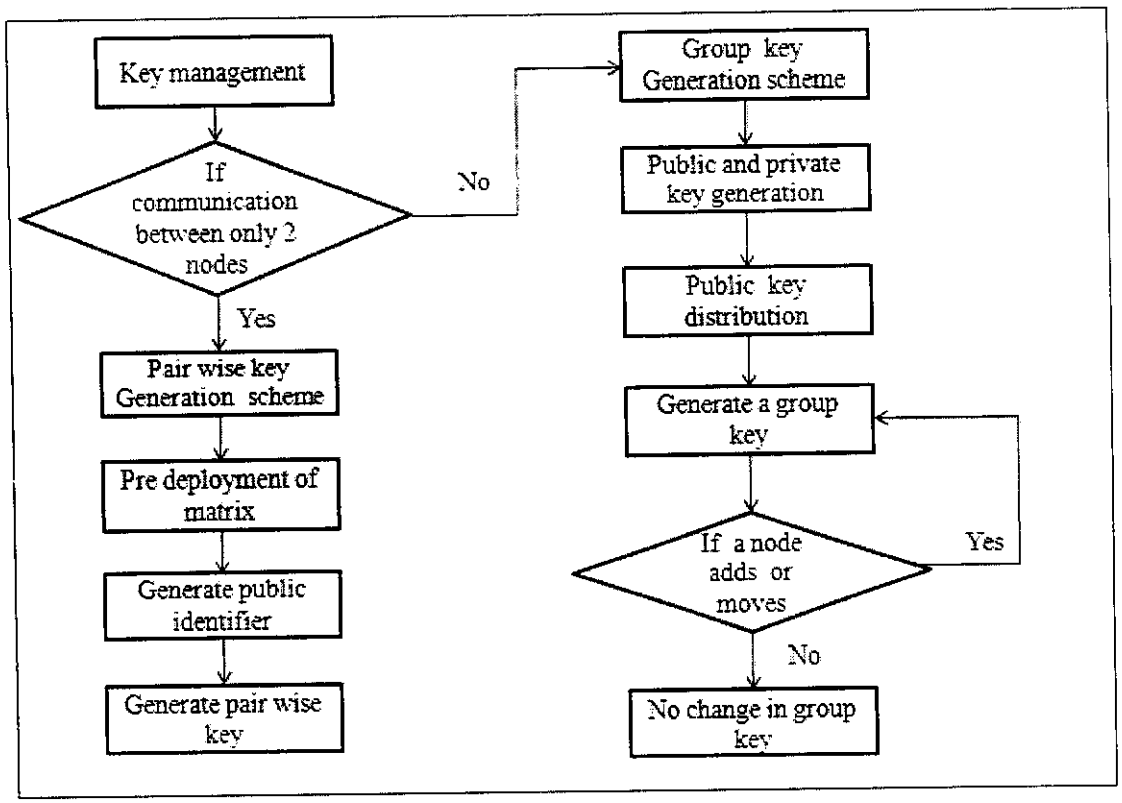


Figure 8: System model

In our system, we consider two cases where communication is between two nodes and in other case the communication is among group of nodes. In both cases we consider a set of keying materials to be pre loaded in nodes. Then the authentication value is generated using hash function. It can be verified and the pair wise keys and group keys are generated. The flow of the model is shown in figure 8 in which two different scenarios are considered.

In first scenario we generate a pair wise key between two nodes in which matrix values are pre-loaded into the nodes. Symmetric matrix, linear independent matrix, random matrix and public identifiers are loaded into the nodes.

Using these values we establish key by verifying authentication value which is calculated by using hash function and by verifying the authentication value we generate a pair wise key by exchanging the public identifiers. The public identifiers and the symmetric matrix is multiplied together to obtain pair wise keys.

3.4 Modules:

- Preloading of keying materials.
- Generation of pair wise keys.
- Generation of group keys.

3.4.1 Algorithm:

1) Pre loading of keying materials that is specified below are loaded into all the nodes in network before their deployment.

   The four matrices that are loaded into the nodes are:

   1.1 Symmetric matrix (A), where values of the matrix above and below the diagonal are similar.

   1.2 Linear independent matrix (B), where the value of each row and column does not depend on the other values of rows and columns.

   1.3 Random matrix(R) which consists of a column of random values.

   1.4 Public identifiers which is a column vector.

2) Key establishment by nodes:

   2.1 Compute F= (A*B), F1=F'

   2.2 Final secret matrix S=F1*B

   2.3 R1=sum(R), where R is a random matrix.R1 is obtained by adding the values of matrix R.

   2.4 Hash function of (R1), where R1 is a resultant matrix value. To the resultant value hash function is applied to obtain the authentication value. The hash function involves certain ex-or and left shift operation.

   2.5 Authentication value is verified by checking the hash value for the neighbor nodes.

   2.6 If authenticated then pair wise keys are generated in a finite Galois field GF (q).

   (S*I1)'*I2, (S*I2)'*I1

3) Group key generation:

3.1 Symmetric matrix (A), linear independent matrix (B) and random matrix(R) are multiplied to generate a resultant matrix.

Thus resultant matrix=A*B*R.

3.2 Resultant matrix values are added and hash function is applied to calculate authentication value, which is given by Hash (sum (resultant matrix value)).

3.3 After authentication group keys are generated using resultant value and hash value, which is given by product of sum of resultant value and hash value.

Group key = sum (resultant value)* hash value

3.4 If any node leaves, new random matrix is generated by group controller node and sent along with the hello message and rekeying is done. Using the new random matrix new resultant matrix is generated and in turn a new group key will be generated.

3.4.2 Overall description:

Initially we generate a secret symmetric matrix (A), of any number of rows and columns. This matrix is pre deployed to all the nodes that come into the network. The direct relation between the secret matrix and the public identifier is changed and we introduce another linear independent matrix(B).These two matrix are multiplied and transpose is taken for the resultant matrix and multiplied again with the linear independent matrix to generate a second level symmetric matrix. Each node is given a public identifier as matrix. To generate pair wise keys between any pair of nodes they exchange their public identifiers initially and it is multiplied with the second level symmetric matrix. Then nodes are verified for authentication where a random matrix is used initially and sum of random matrix is calculated and hash function is applied to obtain the authentication value and if they satisfy the authentication conditions then the secure pair wise keys are generated and the nodes can communicate with one another.

In case of group keys, we initially provide the matrix values. Then all these three matrix values are multiplied and the resultant matrix is applied hash function to generate the authentication value.

The generated authentication value and the resultant matrix are used to generate the group keys. If any node moves out of the network a new random matrix is generated by the group controller node and sent along with hello message. Then new group key is calculated accordingly. If a new node comes into the network its authentication is verified and group key is generated.

3.4.3 Pre-loading of keys:

Initially the network consists of large number of nodes. We consider the nodes to be tamper proof that is it cannot be corrupted. Each node is preloaded with certain keying materials. We provide a symmetric matrix where the values of all the rows and columns are similar above the diagonal values. Then we pre-load a linear independent matrix where the value of each row or column does not depend upon other rows or columns of the same matrix. A random matrix is generated and it is also deployed in the node. Each node is given an identifier which is kept public. All these values are preloaded into the node.

3.4.4 Key establishment:

Using the symmetric matrix and the linear independent matrix we first generate a matrix which is a product of these two matrixes. The resultant value is multiplied again with the linear independent matrix to generate a secret second level symmetric matrix. Then secret second level symmetric matrix and the public identifiers are multiplied to generate the pair wise key. This pair wise key is computed for any number of public identifiers. All these computation is performed by the node itself. Before generating the key, authentication value is generated by using the random matrix. A hash function is applied to the random matrix to generate authentication value. After authentication is verified pair wise key is established between the nodes.

3.4.5 Group key generation:

If a group of nodes want to communicate with one another, then we generate group key. Here symmetric matrix, linear independent matrix and random matrix is pre-loaded into nodes and by multiplying all these values we generate a resultant matrix, which is given by the formula Resultant matrix=A*B*R. The values of the resultant matrix are added to obtain a single value. Then hash function is applied to that single value and we obtain an authentication value and authentication value= hash function (sum (resultant matrix)). This authentication value and the resultant matrix sum value are multiplied to generate the group key. Before the generation of group key, we verify for the authentication.

Whenever a node joins the network the group controller node checks for authentication and group key is given for that node. If a node leaves a network, a new random matrix is generated and it is sent along with a hello message by the group controller node.

3.6 System Specification

3.6.1 Hardware Requirement:

Processor	: Pentium III
Clock speed	: 550MHz
Hard Disk	: 20GB
RAM	: 128MB
Cache Memory	: 512KB
Operating System	: Windows XP
Monitor	: Color Monitor
Keyboard	: 104Keys
Mouse	: 3Buttons

3.6.2 Software Requirement:

Tool	: Net Beans
Language	: Java

CHAPTER 4
TECHNOLOGY INFRASTRUCTURE

4.1 Net Beans:

The newest features in Net Beans 6.7 involve the integration of Project Kenai - a collaborative environment for developers to host their open-source projects - native Maven support and Hudson integration. This release also offers enhancements for Java, PHP, Ruby, JavaScript, Groovy and C/C++, among others, and brings support for SVG Rich Components, remote debugging in Ruby and the latest version of GlassFish. Update 6.7.1 is also available. It is considered a minor update although it does add support for the latest features of JavaFX with JavaFX SDK 1.2 being bundled with the IDE. It also incorporates bug fixes included in Patch 1 for the Net Beans 6.7 release.

4.1.1 Features & Benefits:

- Integration with Project Kenai: New 'Team' menu allows developers to view and create Kenai projects, get source code, chat with other developers from within the Net Beans IDE.
- Native Maven support: With a POM editor and visual library dependency viewer, this feature allows users to build large projects with Maven, manage the POM file, identify and fix library version conflicts easily - all done without wrappers or slowdowns.
- Improved Java, JavaScript, PHP, CSS, support : Code completion, real-time error checking, debugging for Java, JavaScript, PHP, HTML, CSS for development simplified with an integrated, intelligent editor.

4.1.2 System Requirements:

The Net Beans IDE (minimum screen resolution is 1024 x 768 pixels) runs on operating systems that support the Java Virtual Machine (VM) and has been tested on the following platforms:

4.1.3 Minimum hardware configurations:

Microsoft Windows XP Professional SP3:

- Processor: 800MHz Intel Pentium III or equivalent
- Memory: 512 MB
- Disk space: 750 MB of free disk space

Net Beans IDE 6.7.1 is a minor update to Net Beans IDE 6.7 and includes the following changes:

- Availability of the latest IDE bundled with JavaFX SDK 1.2.1
- Numerous community nominated bug fixes included in Patch 1 for Net Beans IDE 6.7

The following updates to release 6.5.1 included in version 6.7 also apply to 6.7.1:

- Maven support for the creation of plug-in and web services as well as support for POM and J2EE
- Kenai integration enables the creation and editing of Kenai hosted projects from within the IDE
- PHP improvements include Selenium support and SQL code completion
- C++ support for profiling, Qt library, code refactoring and macro expansion
- Web API Gateway plug-in enables consumption of Web APIs in applications
- Java ME support for CDC projects in the bundled Java ME SDK 3.0

4.1.4 Multilingual Support Information:

Net Beans IDE is available in the following Sun supported translations: Brazilian Portuguese (BR), Japanese (JP) and Simplified Chinese (ZH). Community translations of the IDE are also available in several additional languages and can be downloaded from the Community Contributed section of the IDE Language drop-down menu.

Net Beans IDE is also known to run on the following platforms:

- Open Solaris 2008.11
- Java Desktop System 2
- Microsoft Windows 2000 Professional SP4
- Mac OS X 10.4.11 Intel/Power PC
- Various other Linux distributions, such as Ubuntu 8.x, Red Hat Enterprise Linux and many others.

4.1.5 Required Software:

Net Beans IDE runs on the Java SE Development Kit (JDK) which consists of the Java Runtime Environment plus developer tools for compiling, debugging, and running applications written in the Java language.

The tested JDKs for this release are:

- JDK 6 Update 14
- JDK 5 Update 19

4.2 Features and Tools:

The Net Beans IDE has many features and tools for each of the Java platforms. Those in the following list are not limited to the Java SE platform but are useful for building, debugging, and deploying applications and applets:

Source Code Editor:

- Syntax highlighting for Java, JavaScript, XML, HTML, CSS, JSP, IDL
- Customizable fonts, colors, and keyboard shortcuts
- Live parsing and error marking
- Pop-up Javadoc for quick access to documentation
- Advanced code completion
- Automatic indentation, which is customizable
- Word matching with the same initial prefixes
- Navigation of current class and commonly used features
- Macros and abbreviations
- Goto declaration and Goto class
- Matching brace highlighting
- JumpList allows you to return the cursor to previous modification

GUI Builder:

- Fully WYSIWYG designer with Test Form feature
- Support for visual and non visual forms
- Extensible Component Palette with preinstalled Swing and AWT components
- Component Inspector showing a component's tree and properties
- Automatic one-way code generation, fully customizable
- Support for AWT/Swing layout managers, drag-and-drop layout customization
- Powerful visual editor
- Support for null layout
- In-place editing of text labels of components, such as labels, buttons, and text fields
- JavaBeans support, including installing, using, and customizing properties, events, and customizers
- Visual Java Bean customization -- ability to create forms from any JavaBean classes
- Connecting beans using Connection wizard

Database Support:

- Database schema browsing to see the tables, views, and stored procedures defined in a database
- Database schema editing using wizards
- Data view to see data stored in tables
- SQL and DDL command execution to help you write and execute more complicated SQL or DDL commands
- Migration of table definitions across databases from different vendors
- Works with databases, such as MySQL, PostgreSQL, Oracle, IBM DB2, Microsoft SQL Server, PointBase, Sybase, Informix, Cloudscape, Derby, and more

The Net Beans IDE also provides full-featured refactoring tools, which allow you to rename and move classes, fields, and methods, as well as change method parameters. In addition, you get a debugger and an Ant-based project system.

4.3 Familiar with Net Beans:

To get started, download the latest stable version from the NetBeans.org web site and install it on whatever platform you use for programming and development. The NetBeans.org web site lists the computer requirements needed to run the IDE.

Net Beans can automatically upgrade its core and extension modules over the Internet, and it has a module that runs periodically on your behalf to check for updates to the version of Net Beans you're using. In addition, the *Update Center* can update and install any modules you okay or request. When you start the application, you should get a welcome screen similar to Figure 9.You'll notice that this welcome screen makes getting started right away an easy process by displaying a Quick Start Guide up front, as well as the options to begin a project or open one. In addition, you can select the Sample Project for a quick example of how code is set up in this IDE. Net Beans is fairly intuitive to use, especially if you've used IDE software in the past.

You'll get familiar with three concepts right away: *projects*, *nodes*, and *workspaces*. Within Net Beans, you work within the context of a project, which consists of an organized group of source files and associated metadata; project-specific properties files; an Ant build script and run settings; and all the tools you'll need to write, compile, test, and debug your application. You can create a main project with subprojects, and you can link projects through dependencies. So getting started is as easy as giving your project a name.

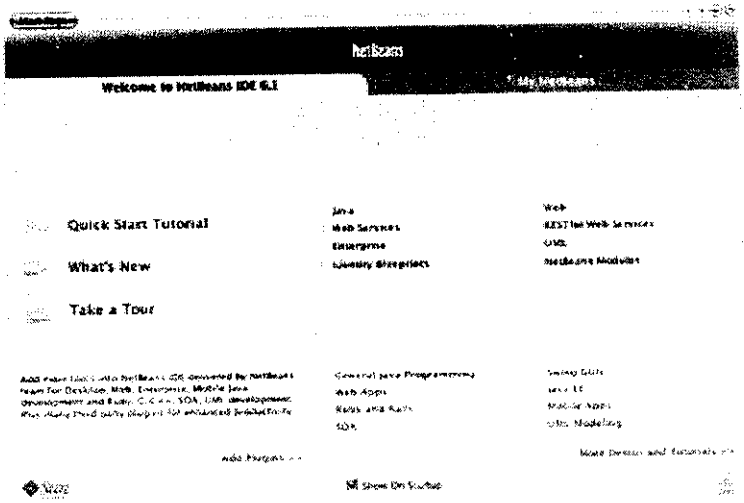


Figure 9: Welcome Screen

Once you tell Net Beans the name of a new project, it then:

- Creates a source tree with an optional skeleton class inside
- Creates a folder for unit tests
- Sets classpaths for compiling, running, and testing
- Sets the Java platform the project is to run on
- Creates an Ant build script (build.xml), which contains instructions that the IDE uses when you perform commands on your project, such as compile or run

Click File from the main menu and select New Project. The New Project wizard pops up and looks similar to Figure 10.

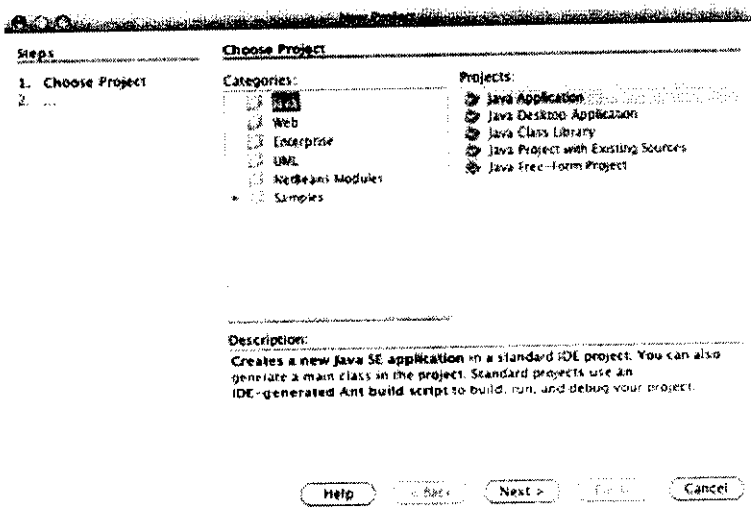


Figure 10: New Project Wizard

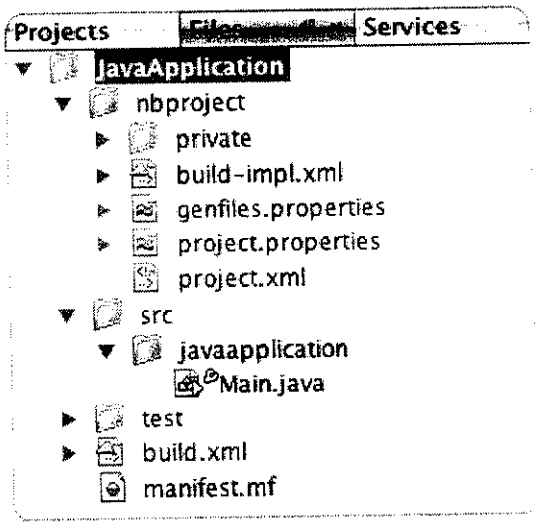


Figure 11: Files Window

From there, you can choose the type of application you want to create by selecting a category such as Java, Web, or Enterprise. Notice that within the Java category, you can create a project containing an empty application. Choose Java and select Java Application. Next, name the project and select a location within your file system. Note that the IDE automatically creates a main class for your application if you want it to. Click Finish.

The Projects window displays only the files that are likely to be regularly edited, such as source files and tests. To see more details about your project, click on the Files tab. The Files tab shows a directory-based view of your project, as shown in Figure 12 above. If you click on the project folder you created, you'll see that the folder contains the Ant script and properties files that control how your project is built and run.

The Files folder also contains the output folder for compiled classes, JAR files (for Java SE projects) or WAR files (for web projects), and Javadoc. This displays as a `build` folder after you compile the application.

Net Beans allows you to see all your objects in a project represented as nodes of a tree, each having its own icon to represent the type of object the node represents. Within the Files tab, you can easily view the trees and representative nodes. If double-click on a node, it opens up into a sub tree that contains more detail. You can collapse or expand trees as necessary. Right-clicking on any node provides easy access to specific functions that you can perform and tools that you can use on that object. Expand the sub trees of the project node that you just created, and you will notice that the fields, constructors, methods, and bean patterns appear as node branches.

CHAPTER 5

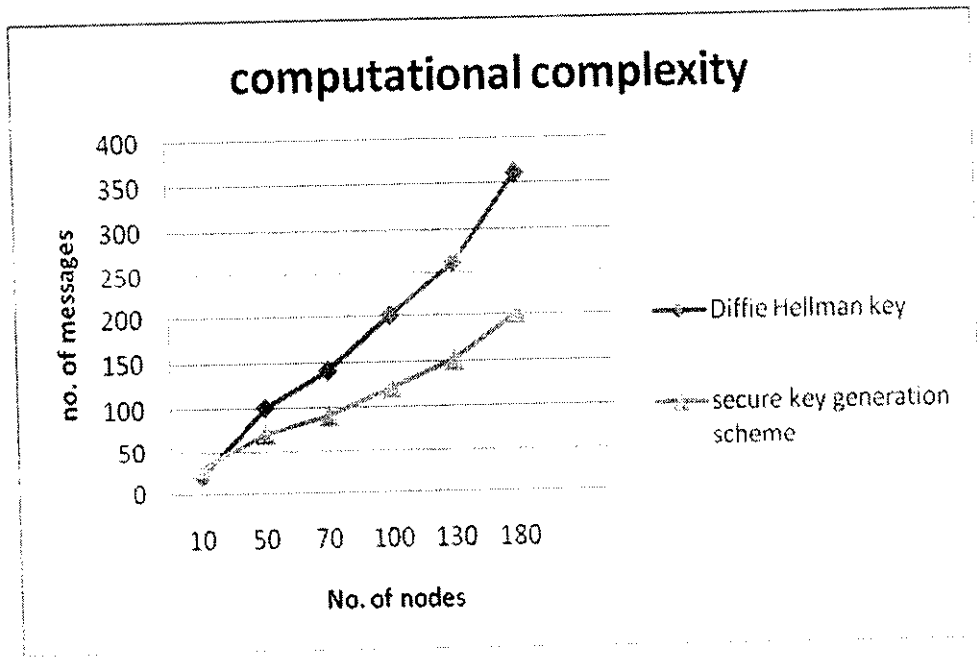
EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Computational complexity:

Diffie Hellman key exchange is implemented and group keys are generated initially. The number of messages required to generate group keys is given by (2N+1) Where N is the number of nodes in the network. In Diffie Hellman key exchange method more number of computations are involved and it requires lot of messages to be shared, so that they can generate the group key. In Secure key generation scheme we used some less number of vector matrix where the number of messages is reduced to some extent. To calculate the number of messages in proposed scheme we use the formula 3m+N, where m is the number of vector columns used in the key computation.

No. of nodes N	Computational complexity	
	DH (2N+1)	SKGS (3m+N),m=7
10	21	31
50	101	71
70	141	91
100	201	121
130	261	151
150	301	171

Table 2: Computational complexity comparison

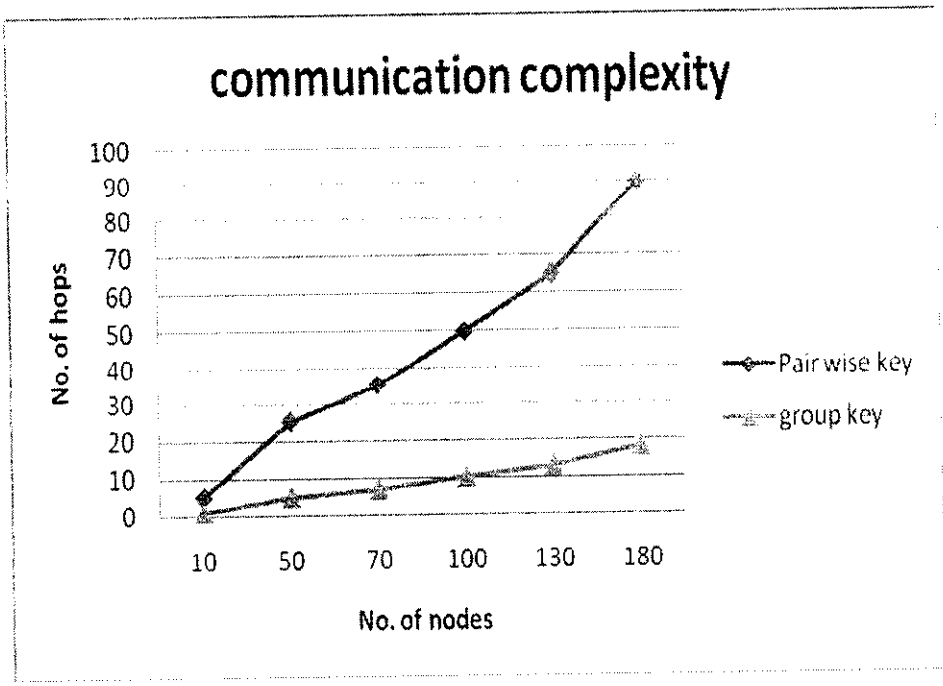


5.2 Communicational complexity:

Here we calculate the number of hops that is needed to communicate the information. In case of pair wise keys each two nodes can calculate a secret key to communicate with one another. Each link key is unique. Hence the total number of nodes is divided by half to generate the pair wise keys. It needs N/2 hops to communicate between a source and destination. In case of group keys we assume that ten nodes form a group and hence among the ten nodes within one hop they can communicate with each other. If the number of nodes increases, then the number of hops also increases. It is given by the formula N/10.

No. of nodes N	Communication complexity	
	Pair wise key (N/2)	Group key (N/10)
10	5	1
50	25	5
70	35	7
100	50	10
130	65	13
180	90	18

Table 3: communicational complexity comparison

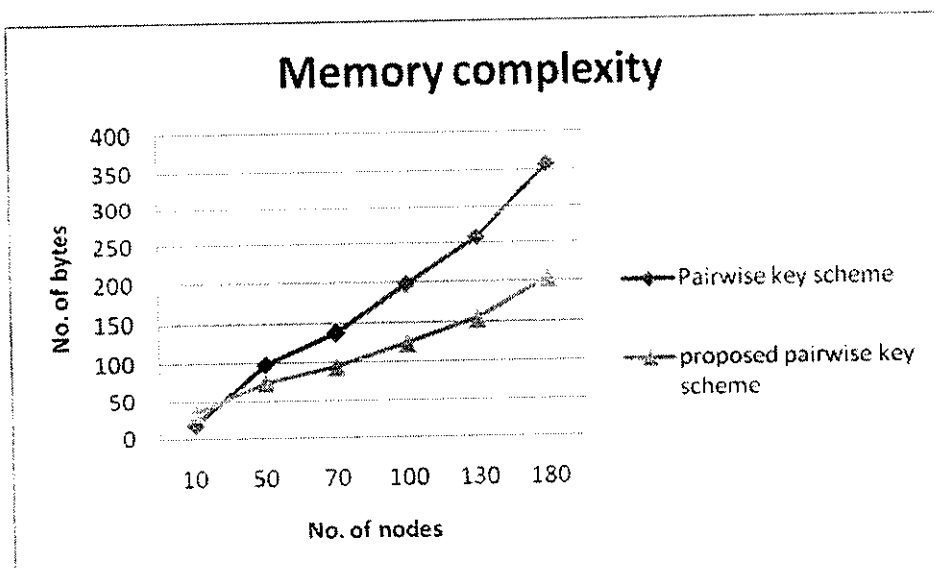


5.3 Memory complexity:

In sensor networks, memory storage is another challenge where it should be used effectively. The amount of memory used is measured in terms of bytes. In pair wise key scheme 2(N-1) bytes of memory space is needed. N is the number of nodes. In proposed pair wise scheme 3m+N space is needed where m=8 is the number of column vectors used in computation of pair wise key.

No. of npdes N	Memory complexity	
	Pair wise key 2(N-1)	PPK (3m+N),m=8
10	18	34
50	98	74
70	138	94
100	198	124
130	258	154
180	358	204

Table 4: Memory complexity comparison



CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion:

In this project Blom scheme which is the base for generation of pair wise key is worked and it is not much secured due to linear dependency among the value. Hence I generate linearly independent values to calculate pair wise keys between the nodes, if communication is between two nodes. If communication is among a group of nodes we generate a group key. Here extension of Diffie Hellman key exchange is done and in the proposed scheme to provide security authentication value is generated. After the verification of authentication value pair wise and group keys are established. The proposed work apart from providing security it also reduces the computational and memory complexity to some extent.

6.2 Future Work:

In future work, nodes can be created and keys can be loaded and established in real time. To select a group controller node in group key scheme a selection algorithm can be included. Rekeying in group key scheme may reduce node battery life time due to more computations, instead of doing calculation every time the node movement occurs, it can be done in batch such that security should also be increased.

APPENDIX I

SOURCE CODE

```
package javaapplication2;
import java.io.*;
import java.util.*;
import java.util.Timer.*;
import java.util.TimerTask.*;
public class Main {
   public static void main(String[] args) {
      DataInputStream in=new  DataInputStream(System.in);
 int i,r,c,j,r1,c1,k,x,y,R,C;
int [][]s=new int[4][4];
int [][]l=new int[4][4];
int [][]m=new int[4][4];
int [][]m1=new int[4][4];
int [][]m2=new int[4][4];
int [][]ra=new int[4][4];
int [][]re=new int[4][4];
int [][]u1=new int[4][4];
int [][]u2=new int[4][4];
int [][]u3=new int[4][4];
int [][]u4=new int[4][4];
int [][]u5=new int[4][4];
int [][]u6=new int[4][4];
int [][]u7=new int[4][4];
int [][]u11=new int[4][4];
int [][]u12=new int[4][4];
int [][]u22=new int[4][4];
int [][]u21=new int[4][4];
int [][]u33=new int[4][4];
int [][]u31=new int[4][4];
int[][]u44=new int[4][4];
int [][]u41=new int[4][4];
int[][]u55=new int[4][4];
int [][]u51=new int[4][4];
int[][]u66=new int[4][4];
int [][]u61=new int[4][4];
int[][]u77=new int[4][4];
int [][]u71=new int[4][4];
int [][]u81=new int[4][4];
int [][]ut=new int[4][4];
int [][]ut1=new int[4][4];
```

```
int [][]ut2=new int [4][4];
int [][]ut3=new int[4][4];
int [][]ut4=new int[4][4];
int [][]ut5=new int[4][4];
int [][]ut6=new int[4][4];
int [][]ut7=new int[4][4];
try {
 System.out.println("enter the no. of rows");
 r=Integer.parseInt(in.readLine());
 System.out.println("enter the no. of columns");
 c=Integer.parseInt(in.readLine());
 System.out.println("enter the symmetric matrix");
  for(i=0;i<r;i++)
 {
 for(j=0;j<c;j++)
 {
s[i][j]=Integer.parseInt(in.readLine());
}
 System.out.println();
 }
System.out.println("SYMMETRIC MATRIX");
for(i=0;i<r;i++)
 {
 for(j=0;j<c;j++)
 {
System.out.print(" "+s[i][j]);
 }
 System.out.println();
}
 System.out.println("enter the no. of rows");
r1=Integer.parseInt(in.readLine());
System.out.println("enter the no. of columns");
c1=Integer.parseInt(in.readLine());
 System.out.println("enter the linearly independent matrix");
 for(i=0;i<r1;i++)
 {
 for(j=0;j<c1;j++)
 {
 l[i][j]=Integer.parseInt(in.readLine());
 }
 System.out.println();
 }
System.out.println("LINEARLY INDEPENDENT MATRIX");
for(i=0;i<r1;i++)
  {
for(j=0;j<c1;j++)
```

```
{
System.out.print(" "+l[i][j]);
 }
System.out.println();
}
 x=s.length;
 y=l.length;
for( i = 0; i <x; i++)
{
 for( j = 0; j <y-1; j++)
{
 for( k = 0; k <y; k++)
{
 m[i][j] += s[i][k]*l[k][j];
}
 }
  }
 System.out.println("Multiply of both matrix : ");
 for( i = 0; i < r1; i++)
{
 for( j = 0; j < c1; j++)
{
  System.out.print(" "+m[i][j]);
  }
 System.out.println();
 }
System.out.println("Transpose of the matrix");
for( i = 0; i < r1; i++)
{
 for( j = 0; j < c1; j++)
{
 System.out.print(" "+m[j][i]);
}
 System.out.println();
 }
 for( i = 0; i < r1; i++)
 {
  for( j = 0; j < c1; j++)
 {
        m2[i][j]=m[j][i];
   }
  }
  for( i = 0; i <x; i++) {
     for( j = 0; j <y-1; j++)
 {
```

```
for( k = 0; k <y; k++)
{
m1[i][j] += m2[i][k]*l[k][j];
 }
}    }
System.out.println("FINAL SYMMETRIC MATRIX : ");
 for( i = 0; i < r1; i++)
{
    for( j = 0; j < c1;j++)
{
System.out.print(" "+m1[i][j]);
 }
 System.out.println();
 }
 System.out.println("Random matrix:");
 System.out.println("enter the no. of rows");
 R=Integer.parseInt(in.readLine());
 System.out.println("enter the no. of columns");
 C=Integer.parseInt(in.readLine());
 System.out.println("Enter the Random matrix");
  for(i=0;i<R;i++)
 {
for(j=0;j<C;j++)
{
 ra[i][j]=Integer.parseInt(in.readLine());
  }
 System.out.println();
}
 System.out.println("RANDOM MATRIX");
    for(i=0;i<R;i++)
    {
  for(j=0;j<C;j++)
 {
 System.out.print(" "+ra[i][j]);


   }
 System.out.println();
}
 int sum=0;
for( i = 0; i <R; i++)
{
 for( j = 0; j <C; j++)
 {
sum=sum+ra[i][j];
   }
```

```
  }
System.out.println("sum of matrix value is:"+sum);
int hash=sum^2;
int hash1=hash<<1;
int hash2=hash1<<2;
int hash3=hash2|3;
System.out.println("group conroller");
System.out.println("hash value calculation");
System.out.println("The final hash value is:"+hash3);
System.out.println("group conroller");
System.out.println("Information of user Identifier");
System.out.println("User 1 info:");
for(i=0;i<3;i++)
{
  for(j=0;j<1;j++)
  {
    u1[i][j]=Integer.parseInt(in.readLine());
  }
  System.out.println();
}
System.out.println("User 2 info:");
for(i=0;i<3;i++)
{
  for(j=0;j<1;j++)
  {
    u2[i][j]=Integer.parseInt(in.readLine());
  }
System.out.println();
}
System.out.println("User 3 info:");
for(i=0;i<3;i++)
{
  for(j=0;j<1;j++)
  {
    u3[i][j]=Integer.parseInt(in.readLine());
  }
}
System.out.println("User 4 info:");
for(i=0;i<3;i++)
{
  for(j=0;j<1;j++)
  {
    u4[i][j]=Integer.parseInt(in.readLine());
  }
}
System.out.println("User 5 info:");
```

```
for(i=0;i<3;i++)
{
   for(j=0;j<1;j++)
   {
      u5[i][j]=Integer.parseInt(in.readLine());
   }
}
System.out.println("User 6 info:");
for(i=0;i<3;i++)
{
   for(j=0;j<1;j++)
   {
     u6[i][j]=Integer.parseInt(in.readLine());
   }
}
for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
u11[i][j] += m1[i][k]*u1[k][j];
}
 }
  }
 for( i = 0; i <1; i++)
{
 for( j = 0; j <c; j++)
{
    ut[i][j]=u11[j][i];
 }
}
 for( i = 0; i <r; i++)
 {
  for( j = 0; j <c; j++)
 {
  for( k = 0; k <c-1; k++)
 {
 u12[i][j] += ut[i][k]*u2[k][j];
 }
  }
   }
  for( i = 0; i <1; i++)
  {
   for( j = 0; j <1; j++)
  {
```

```
System.out.println("pair wise value of node 1 is:"+u12[i][j]);
int p1=u12[i][j]*hash3;
System.out.println("authenticated pair wise value of node1: "+p1);
 }
}
for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u22[i][j] += m1[i][k]*u2[k][j];
}
 }
  }
 for( i = 0; i <1; i++)
 {
 for( j = 0; j <c; j++)
 {
    ut1[i][j]=u22[j][i];
 }}
 for( i = 0; i <r; i++)
 {
 for( j = 0; j <c; j++)
 {
 for( k = 0; k <c-1; k++)
 {
 u21[i][j] += ut1[i][k]*u1[k][j];
 }
 }
  }
 for( i = 0; i <1; i++)
 {
 for( j = 0; j <1; j++)
 {
 System.out.println("pair wise value of node 2 is:"+u21[i][j]);
 int p2=u21[i][j]*hash3;
 System.out.println("authenticated pair wise value of node2: "+p2);
 }
 }
 for( i = 0; i <r; i++)
 {
 for( j = 0; j <c; j++)
 {
 for( k = 0; k <c-1; k++)
 u33[i][j] += m1[i][k]*u3[k][j];
```

```
}
 }
for( i = 0; i <1; i++)
{
 for( j = 0; j <c; j++)
{
    ut2[i][j]=u33[j][i];
 }
}
for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u31[i][j] += ut2[i][k]*u4[k][j];
}
 }
  }
 for( i = 0; i <1; i++)
 {
  for( j = 0; j <1; j++)
 {
 System.out.println("pair  wise value of node 3 is:"+u31[i][j]);
 int p3=u31[i][j]*hash3;
 System.out.println("authenticated pair wise value of node3: "+p3);
  }
 }
 for( i = 0; i <r; i++)
 {
  for( j = 0; j <c; j++)
 {
  for( k = 0; k <c-1; k++)
 {
  u44[i][j] += m1[i][k]*u4[k][j];
 }
  }
   }
  for( i = 0; i <1; i++)
 {
  for( j = 0; j <c; j++)
 {
     ut3[i][j]=u44[j][i];
  }
 }
  for( i = 0; i <r; i++)
```

```
{
for( j = 0; j <c; j++)
{
for( k = 0; k <c-1; k++)
{
u41[i][j] += ut3[i][k]*u3[k][j];
}
 }
  }
for( i = 0; i <1; i++)
{
 for( j = 0; j <1; j++)
{
System.out.println("pair wise value of node 4 is:"+u41[i][j]);
int p4=u41[i][j]*hash3;
System.out.println("authenticated pair wise value of node4: "+p4);
 }
}
for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u55[i][j] += m1[i][k]*u5[k][j];
}
 }
  }
 for( i = 0; i <1; i++)
 {
  for( j = 0; j <c; j++)
 {
     ut4[i][j]=u55[j][i];
  }
 }
 for( i = 0; i <r; i++)
 {
  for( j = 0; j <c; j++)
 {
  for( k = 0; k <c-1; k++)
 {
  u51[i][j] += ut4[i][k]*u6[k][j];
 }
  }
   }
 for( i = 0; i <1; i++)
```

```
{
 for( j = 0; j <1; j++)
{
System.out.println("pair wise value of node 5 is:"+u51[i][j]);
int p5=u51[i][j]*hash3;
System.out.println("authenticated pair wise value of node5: "+p5);
 }
}
 for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u66[i][j] += m1[i][k]*u6[k][j];
}
 }
  }
for( i = 0; i <1; i++)
{
 for( j = 0; j <c; j++)
{
    ut5[i][j]=u66[j][i];
 }
}
for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u61[i][j] += ut5[i][k]*u5[k][j];
}
 }
  }
for( i = 0; i <1; i++)
{
 for( j = 0; j <1; j++)
{
System.out.println("pair wise value of node 6 is:"+u61[i][j]);
int p6=u61[i][j]*hash3;
System.out.println("authenticated pair wise value of node6: "+p6);
 }
}
System.out.println("New node joins..");
System.out.println("User 7 joins:");
```

```
for(i=0;i<3;i++)
{
   for(j=0;j<1;j++)
   {
    u7[i][j]=Integer.parseInt(in.readLine());
   }
}
int hash4=hash3;
if(hash4==hash3)
   System.out.println("authenticated node");
for( i = 0; i <r; i++)
{
for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u77[i][j] += m1[i][k]*u7[k][j];
}
 }
  }
 for( i = 0; i <1; i++)
 {
  for( j = 0; j <c; j++)
  {
     ut6[i][j]=u77[j][i];
  }
 }
 for( i = 0; i <r; i++)
 {
  for( j = 0; j <c; j++)
  {
  for( k = 0; k <c-1; k++)
  {
  u71[i][j] += ut6[i][k]*u6[k][j];
  }
  }
   }
  for( i = 0; i <1; i++)
  {
   for( j = 0; j <1; j++)
   {
  System.out.println("pair  wise value of node 7 is:"+u71[i][j]);
  int p7=u71[i][j]*hash3;
  System.out.println("authenticated pair wise value of node7: "+p7);
   }
  }
```

```
for( i = 0; i <r; i++)
{
 for( j = 0; j <c; j++)
{
 for( k = 0; k <c-1; k++)
{
 u81[i][j] += ut5[i][k]*u7[k][j];
}
 }
  }
for( i = 0; i <1; i++)
{
 for( j = 0; j <1; j++)
{
System.out.println("pair  wise value of node 6 is:"+u81[i][j]);
int p61=u81[i][j]*hash3;
System.out.println("authenticated pair wise value of node6: "+p61);
 }
}
System.out.println("If a node leaves..");
System.out.println("New authentication value");
System.out.println("New Random matrix:");
 System.out.println("enter the no. of rows");
 R=Integer.parseInt(in.readLine());
 System.out.println("enter the no. of columns");
 C=Integer.parseInt(in.readLine());
 System.out.println("Enter the Random matrix");
   for(i=0;i<R;i++)
 {
for(j=0;j<C;j++)
{
ra[i][j]=Integer.parseInt(in.readLine());   }
 System.out.println();
}
 System.out.println("NEW RANDOM MATRIX");
    for(i=0;i<R;i++)
    {
   for(j=0;j<C;j++)
 {
 System.out.print(" "+ra[i][j]);
    }
 System.out.println();
   }
 int Sum=0;
for( i = 0; i <R; i++)
{
```

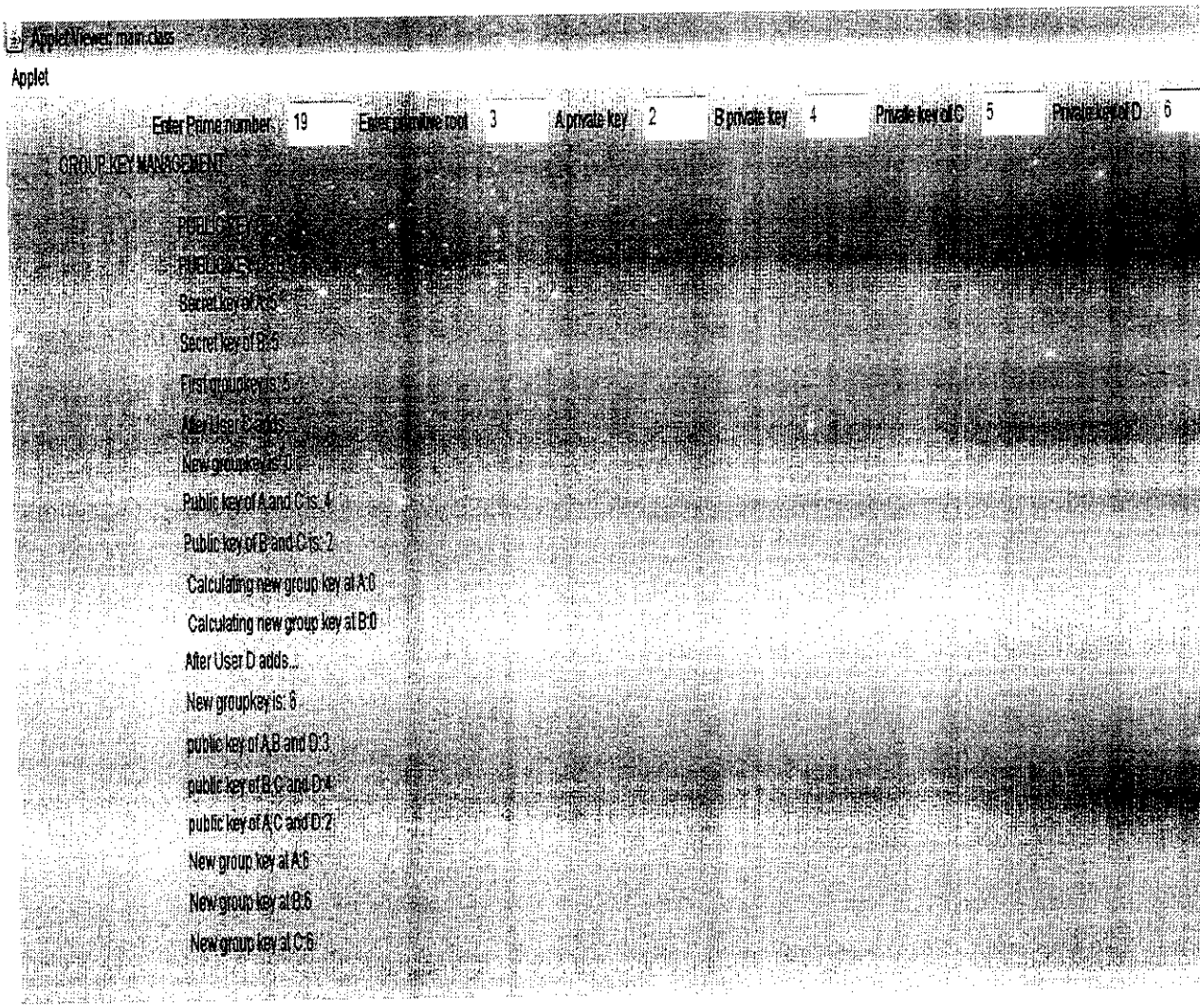
```
for( j = 0; j <C; j++)
{
Sum=Sum+ra[i][j];
  }
   }
System.out.println("sum of matrix value is:"+Sum);
int Hash=Sum^2;
int Hash1=Hash<<1;
int Hash2=Hash1<<2;
int Hash3=Hash2|3;
System.out.println("New hash value is:"+Hash3);
for( i = 0; i <1; i++)
{
 for( j = 0; j <1; j++)
{
int p1=u12[i][j]*Hash3;
System.out.println("authenticated pair wise value of node1: "+p1);
 }
}
for( i = 0; i <1; i++)
{
 for( j = 0; j <1; j++)
{
int p2=u21[i][j]*Hash3;
System.out.println("authenticated pair wise value of node2: "+p2);
 }
}
for( i = 0; i <1; i++)
{
 for( j = 0; j <1; j++)
{
int p3=u31[i][j]*Hash3;
System.out.println("authenticated pair wise value of node3: "+p3);
 }
}
for( i = 0; i <1; i++)
{
for( j = 0; j <1; j++)
{
int p4=u41[i][j]*Hash3;
System.out.println("authenticated pair wise value of node4: "+p4);
 }
}
for( i = 0; i <1; i++)
{
for( j = 0; j <1; j++)
```

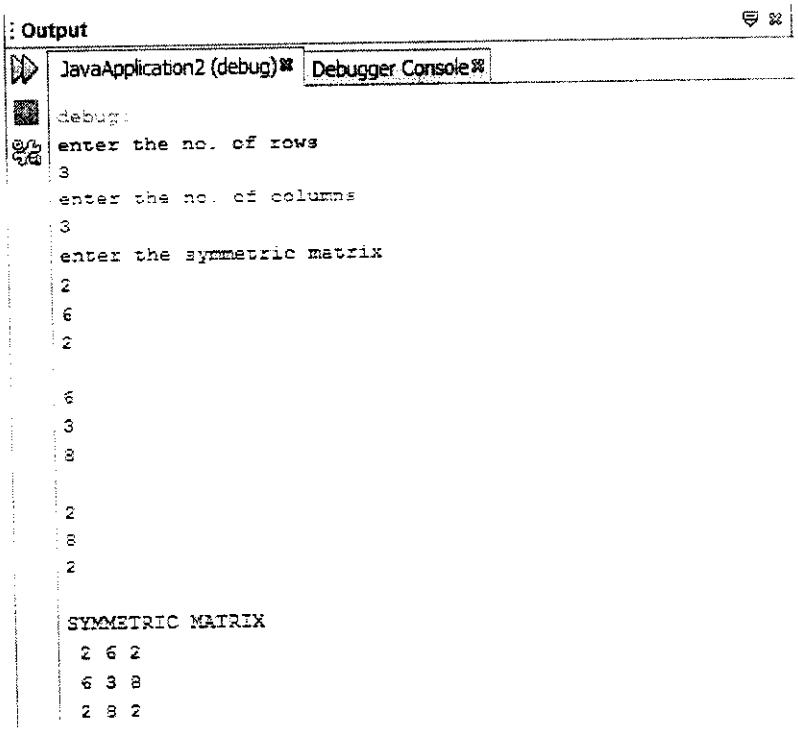
```
{
int p5=u51[i][j]*Hash3;
System.out.println("authenticated pair wise value of node5: "+p5);
 }
}
for( i = 0; i <1; i++)
{
for( j = 0; j <1; j++)
{
int p6=u61[i][j]*Hash3;
System.out.println("authenticated pair wise value of node6: "+p6);
 }
}
int delay = 5000;   // delay for 5 sec.
int period = 100000;  // repeat every sec.
Timer timer = new Timer();
timer.scheduleAtFixedRate(new TimerTask() {
      public void run() {
         try{
         int i,j,R=3,C=1;
         int ra[][]=new int[4][4];
          DataInputStream in=new  DataInputStream(System.in);
         System.out.println("enter a new random no.");
           for(i=0;i<R;i++)
 {
for(j=0;j<C;j++)
{
   ra[i][j]=Integer.parseInt(in.readLine());
  }
 System.out.println();
}
 System.out.println("RANDOM MATRIX");
    for(i=0;i<R;i++)
    {
  for(j=0;j<C;j++)
 { System.out.print(" "+ra[i][j]);
 }
 System.out.println();
  }
 int sum=0;
for( i = 0; i <R; i++)
{
 for( j = 0; j <C; j++)
 {
sum=sum+ra[i][j];
  }
```

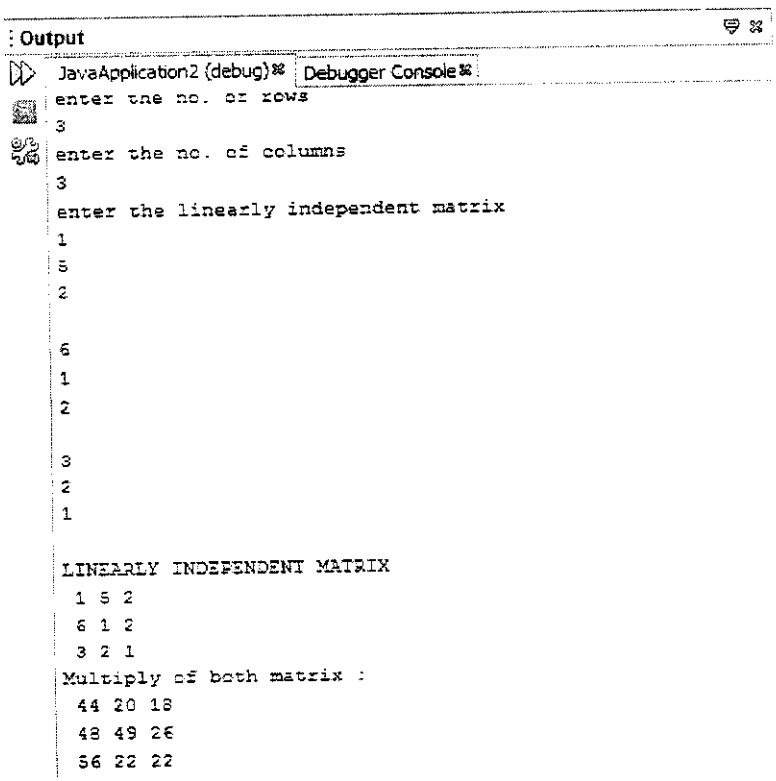
```
  }
System.out.println("sum of matrix value is:"+sum);
int hash=sum^2;
int hash1=hash<<1;
int hash2=hash1<<2;
int hash3=hash2|3;
System.out.println("group conroller");
System.out.println("hash value calculation");
System.out.println("The final hash value is:"+hash3);       }
          catch(Exception e)
 {
 System.out.println(e);
  }
      }
    }, delay, period);
    }
 catch(Exception e)
  {
  System.out.println(e);
   }
    }
 }
```

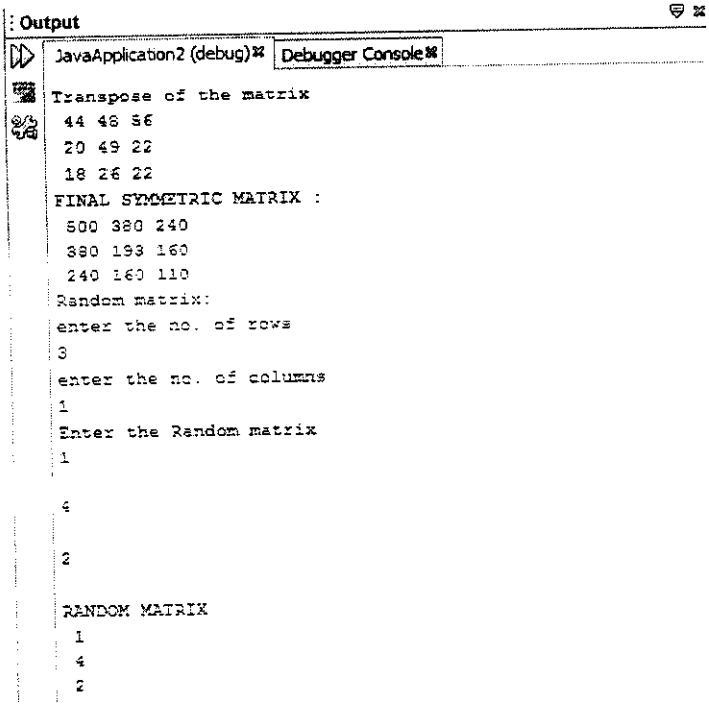
APPENDIX II

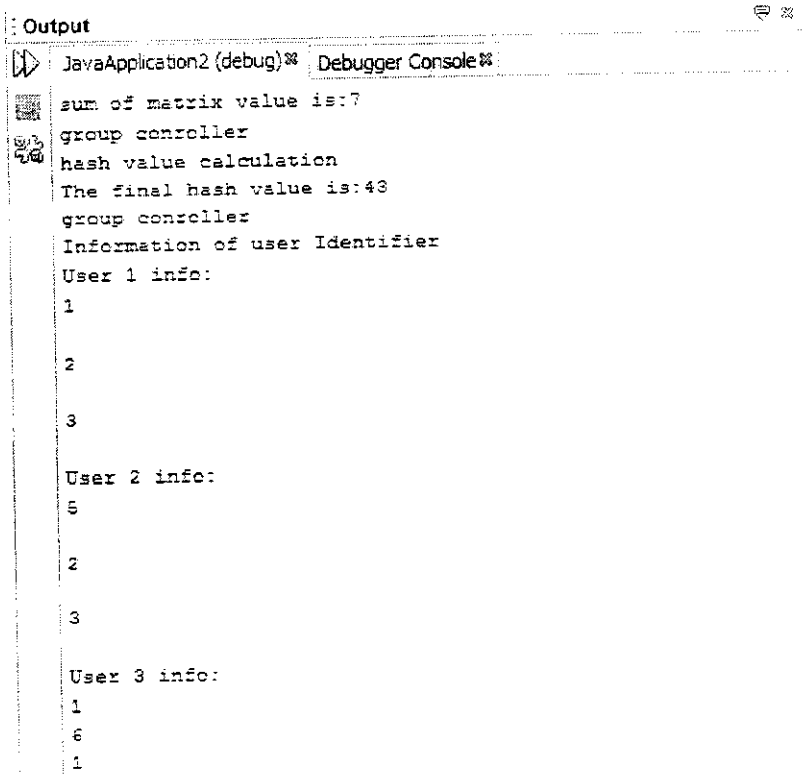
SNAP SHOTS

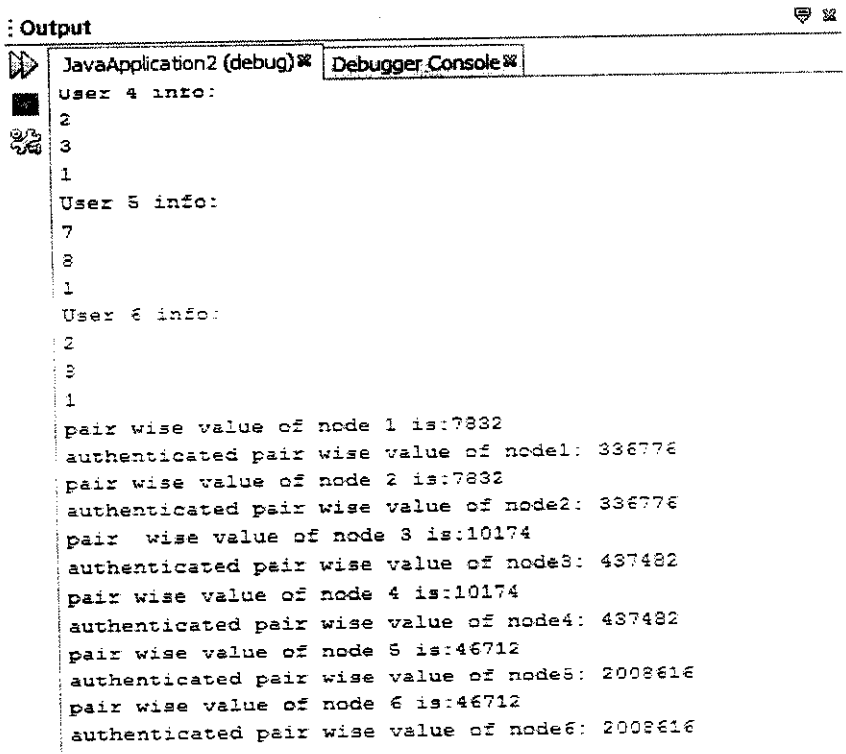


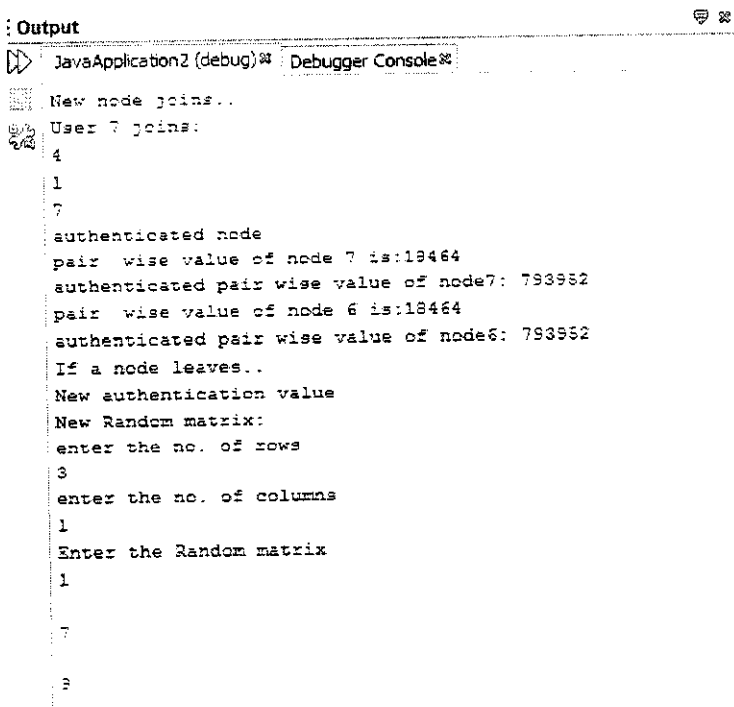

Output
JavaApplication2 (debug)
Debugger Console
debug:
enter the no. of rows
3
enter the no. of columns
3
enter the symmetric matrix
2
6
2
6
3
8
2
8
2
SYMMETRIC MATRIX
2 6 2
6 3 8
2 8 2


Output
JavaApplication2 (debug)
Debugger Console
enter the no. of rows
3
enter the no. of columns
3
enter the linearly independent matrix
1
5
2
6
1
2
3
2
1
LINEARLY INDEPENDENT MATRIX
1 5 2
6 1 2
3 2 1
Multiply of both matrix :
44 20 18
48 49 26
56 22 22


Output
JavaApplication2 (debug)
Debugger Console
Transpose of the matrix
44 48 56
20 49 22
18 26 22
FINAL SYMMETRIC MATRIX :
500 380 240
380 193 160
240 160 110
Random matrix:
enter the no. of rows
3
enter the no. of columns
1
Enter the Random matrix
1
4
2
RANDOM MATRIX
1
4
2


Output
JavaApplication2 (debug)
Debugger Console
sum of matrix value is:7
group controller
hash value calculation
The final hash value is:43
group controller
Information of user Identifier
User 1 info:
1
2
3
User 2 info:
5
2
3
User 3 info:
1
6
1


Output
JavaApplication2 (debug)
Debugger Console
User 4 info:
2
3
1
User 5 info:
7
8
1
User 6 info:
2
3
1
pair wise value of node 1 is:7832
authenticated pair wise value of node1: 336776
pair wise value of node 2 is:7832
authenticated pair wise value of node2: 336776
pair wise value of node 3 is:10174
authenticated pair wise value of node3: 437482
pair wise value of node 4 is:10174
authenticated pair wise value of node4: 437482
pair wise value of node 5 is:46712
authenticated pair wise value of node5: 2008616
pair wise value of node 6 is:46712
authenticated pair wise value of node6: 2008616


Output
JavaApplication2 (debug)
Debugger Console
New node joins..
User 7 joins:
4
1
7
authenticated node
pair wise value of node 7 is:18464
authenticated pair wise value of node7: 793952
pair wise value of node 6 is:18464
authenticated pair wise value of node6: 793952
If a node leaves..
New authentication value
New Random matrix:
enter the no. of rows
3
enter the no. of columns
1
Enter the Random matrix
1
7
3

```
sum of matrix value is:17
New hash value is:155
authenticated pair wise value of node1: 1213960
authenticated pair wise value of node2: 1213960
authenticated pair wise value of node3: 1576970
authenticated pair wise value of node4: 1576970
authenticated pair wise value of node5: 7240360
authenticated pair wise value of node6: 7240360
enter a new random no.
1

5

4

RANDOM MATRIX
 1
 5
 4
sum of matrix value is:10
group controller
hash value calculation
The final hash value is:67
BUILD SUCCESSFUL (total time: 1 minute 45 seconds)
```

Output

JavaApplication4 (debug) | Debugger Console

```
debug:
enter the no. of rows
3
enter the no. of columns
3
enter the symmetric matrix
2
6
2

6
3
8

2
8
2

SYMMETRIC MATRIX
 2 6 2
 6 3 8
 2 8 2
```

Output

JavaApplication4 (debug) | Debugger Console

```
enter the no. of rows
3
enter the no. of columns
3
enter the linearly independent matrix
1
2
3

2
1
2

4
1
3

LINEARLY INDEPENDENT MATRIX
 1 2 3
 2 1 2
 4 1 3
Multiply of both matrix :
 22 12 24
 44 23 48
 26 14 28
```

Output

JavaApplication4 (debug) | Debugger Console

```
Random matrix:
enter the no. of rows
3
enter the no. of columns
1
Enter the Random matrix
1

3

4

RANDOM MATRIX
 1
 3
 4
Resultant matrix
 154
 305
 180
sum of matrix value is:639
```

Output

JavaApplication4 (debug) | Debugger Console

```
Group head
hash value calculation
The final authorization value is:5099
Group key value is:3258261
key value in node1 is:3258261
key value in node2 is:3258261
key value in node3 is:3258261
key value in node4 is:3258261
If node4 leaves...
Random matrix:
enter the no. of rows
3
enter the no. of columns
1
Enter the Random matrix
4

5

2



RANDOM MATRIX
 4
 5
 2
sum of matrix value is:1452
Group head
hash value calculation
The final authorization value is:11635
Group key value is:16894020
key value in node1 is:16894020
key value in node2 is:16894020
key value in node3 is:16894020
If node5 adds..
check authorization value
The final authorization value is:11635
key value in node5 is:16894020
BUILD SUCCESSFUL (total time: 38 seconds)
```

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci,' A survey on sensor networks'. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[2] X.Brain Zhang, S, Lam and Y.Richard Yang, 'Protocol design for scalable and reliable group rekeying' IEEE transactions on networking, vol.11, no.6, 2003

[3] R. Blom. 'An optimal class of symmetric key generation systems', *Advances in Cryptology: Proceedings of EUROCRYPT 84 (Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, eds.), Lecture Notes in Computer Science, Springer-Verlag,* 209:335–338, 1985.

[4] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M.Yung. 'Perfectly -secure key distribution for dynamic conferences', Lecture Notes in Computer Science, 740:471–486, 1993.

[5] Seyit A.camptee and Bulent yenar (2005),'Key distribution mechanisms for adhoc networks-a survey', department of computer science, Rensselaer polytechnic institute,pages-27.

[6] Yi cheng and Dharma P.Agarwal,'Efficient Pair wise Key Establishment and Management in static wireless sensor networks', IEEE, 2005.

[7] Wenling Du et al. 'A pair wise key pre distribution scheme for wireless sensor networks', ACM transactions, October, 2003.

[8] W. Du, J. Deng, Y. S. Han, S. Chen, and P. K. Varshney ,'A key management scheme for wireless sensor networks using deployment knowledge', In Proceedings of the IEEE INFOCOM 2004, pages 586–597, March 2004.

[9]Mohamed Eltoweissy and Ravi Mukkamala, 'Dynamic key management in adhoc networks', IEEE communication magazine, April 2006, pg: 120-130

[10] Laurent Eschenauer and Virgil D.Gligor 'A key management scheme for distributed sensor networks', 2004.

[11]Dijiang Huang, Manish Mehta and Deep Medhi,'Modeling pair wise key establishment for random key predistribution in large scale sensor networks', ACM transactions on networking, vol no.15, no 5, 2007

[12] Takashi Ito, Hidenori Ohta, Nori Matsuda and Takeshi Yoneda ,'A Key predistribution scheme for secure sensor networks using probability density function of node deployment',ACM transactions,Nov,2007.

[13] van der Merwe, J., Dawoud, D., and McDonald,' A survey on peer-to-peer key management for mobile ad hoc networks', ACM Comput. Surv. 39, 1, Article 1 (April 2007), 45 pages.

[14] Chia-Mu Yu,Chun-Shien Lu and Sy-Yen Kuo(2010)'Non interactive pair wise key establishment for sensor networks' IEEE transactions on information security,vol.5,n0.3,pgs-556 to 569.

[15] Sandro Rafalei and David Hutchison,'A Survey of Key Management for Secure Group Communication', Lancaster University, ACM Computing Surveys, Vol. 35, No. 3, September 2003, pp. 309–329.

[16]W. Zhang, M. Tran, S. Zhu, and G. Cao, 'A random perturbation based scheme for pair wise key establishment in sensor networks,' in Proc. ACMInt. Symp. Mobile Ad Hoc Networking and Computing (MobiHoc), Montreal, QC, Canada, Sep. 9–14, 2007.